Document Number: 837817
Version: 4
Date: April 2020

Continuum Math Libraries

Vector Software User's Manual

Curtiss-Wright
Defense Solutions
20130 Lakeview Center Plaza, Suite 200
Ashburn, VA 20147
(703) 779-7800

Revisions

Rev	Date	Description
1	June 2017	First release.

Copyright Notice

The information in this document is subject to change without notice and should not be construed as a commitment by Curtiss-Wright Controls, Inc. While reasonable precautions have been taken, Curtiss-Wright Controls, Inc. assumes no responsibility for any errors that may appear in this document.
No part of this document may be copied or reproduced without the prior written consent of Curtiss-Wright Controls, Inc.
The proprietary information contained in this document must not be disclosed to others for any purpose, nor used for manufacturing purposes, without written permission of Curtiss- Wright Controls, Inc. The acceptance of this document will be construed as an acceptance of the foregoing condition.
Copyright © 2017 Curtiss-Wright Controls, Inc. All rights reserved.

Trademarks

VxWorks and Wind River Systems are registered trademarks of Wind River Systems, Inc.
Windows is a registered trademark of Microsoft Corporation.
Intel is a registered trademark of Intel Corporation.
Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.
All other brand and product names are trademarks or registered trademarks of their respective owners.

Contents

1 Introduction **1-1**
1.1 Purpose of This Manual . . . 1-1
1.2 Quick Reference Guide . . . 1-1
1.3 Full Reference Guide. . . . 1-1
1.4 The Vector API . . . 1-2
1.5 Storage Assumptions . . . 1-3
1.6 Sample Increments . . . 1-3
1.7 Standard Restrictions . . . 1-4

2 Vector Library Overview **2-1**
2.1 Interleaved Complex Functions . . . 2-2
2.2 Complex Split Functions . . . 2-4
2.3 Real Vector Functions. . . . 2-5
2.4 Integer Vector Functions. . . . 2-9

3 Getting the Best Performance **3-1**
3.1 Memory Alignment . . . 3-1
3.2 Vector/Matrix Format . . . 3-2
3.3 Error Checking and Debugging . . . 3-2
3.4 Scalar Functions . . . 3-2
3.5 Random Number Generation . . . 3-3
3.6 Vector and Elementwise Operations . . . 3-3
3.7 Signal Processing Functions . . . 3-3
3.8 FFT Functions . . . 3-3
3.9 FIR Filter, Convolution and Correlation Functions . . . 3-3
3.10 Linear Algebra Functions . . . 3-3
3.11 Matrix and Vector Operations . . . 3-3
3.12 Controlling the Number of Threads . . . 3-4
3.13 Support Functions . . . 3-4

4 Vector Library Function Reference **4-1**
4.1 acorfP . . . 4-1
4.2 acortP . . . 4-3
4.3 aspecP . . . 4-4
4.4 avcfftf . . . 4-6
4.5 avcffti . . . 4-7
4.6 avcft2fc . . . 4-8
4.7 avcft2fr . . . 4-9
4.8 avcft2ic . . . 4-10
4.9 avcft2ir . . . 4-11
4.10 avfftwts . . . 4-12
4.11 avfreewts . . . 4-13
4.12 avrfftf . . . 4-14
4.13 avrffti . . . 4-16
4.14 avrft2fc . . . 4-18
4.15 avrft2fr . . . 4-20
4.16 avrft2ic . . . 4-22

4.17 avrft2ir . . . 4-24
4.18 avtcft2fc . . . 4-26
4.19 avtcft2fr . . . 4-28
4.20 avtcft2ic . . . 4-30
4.21 avtcft2ir . . . 4-32
4.22 avtrft2fc . . . 4-34
4.23 avtrft2fr . . . 4-36
4.24 avtrft2ic . . . 4-38
4.25 avtrft2ir . . . 4-40
4.26 blkmanP . . . 4-42
4.27 blkman-harris . . . 4-43
4.28 ccdotpP . . . 4-44
4.29 cconvdP . . . 4-46
4.30 ccorfP . . . 4-48
4.31 ccortP . . . 4-49
4.32 cdotprP . . . 4-50
4.33 cfftP . . . 4-52
4.34 cfftfP . . . 4-53
4.35 cfftfnP . . . 4-55
4.36 cfftiP . . . 4-57
4.37 cfftinP . . . 4-59
4.38 cfftscP . . . 4-61
4.39 cft2fcP . . . 4-62
4.40 cft2frP . . . 4-64
4.41 cft2icP . . . 4-66
4.42 cft2irP . . . 4-68
4.43 convdP . . . 4-70
4.44 cpows . . . 4-72
4.45 crvdivP . . . 4-74
4.46 crvmulP . . . 4-76
4.47 cspecP . . . 4-78
4.48 csve . . . 4-80
4.49 cvabsP . . . 4-81
4.50 cvabserr . . . 4-83
4.51 cvaddP . . . 4-84
4.52 cvaddsub . . . 4-86
4.53 cvavg . . . 4-88
4.54 cvavgnegy . . . 4-89
4.55 cvaxyplusz . . . 4-90
4.56 cvceil . . . 4-91
4.57 cvclip . . . 4-92
4.58 cvcliph . . . 4-93
4.59 cvclipl . . . 4-94
4.60 cvcmlP . . . 4-95
4.61 cvcmlaP . . . 4-97
4.62 cvcmpXX . . . 4-99
4.63 cvcombP . . . 4-102
4.64 cvconjP . . . 4-104
4.65 cvcsmlP . . . 4-106
4.66 cvcvrsmlP . . . 4-108
4.67 cvdivP . . . 4-110
4.68 cvexpP . . . 4-112
4.69 cvfillP . . . 4-114
4.70 cvfindlastXX . . . 4-116
4.71 cvfindXX . . . 4-119
4.72 cvfloor . . . 4-122
4.73 cvintegrate . . . 4-123

4.74 cvmaP . . . 4-124
4.75 cvmagsP . . . 4-126
4.76 cvmerge . . . 4-128
4.77 cvmexpP . . . 4-129
4.78 cvmgsaP . . . 4-131
4.79 cvmlP . . . 4-133
4.80 cvmlaP . . . 4-135
4.81 cvmovP . . . 4-137
4.82 cvmulP . . . 4-139
4.83 cvnegP . . . 4-141
4.84 cvolsave . . . 4-143
4.85 cvphasP . . . 4-145
4.86 cvrcipP . . . 4-147
4.87 cvrealP . . . 4-149
4.88 cvrelerr . . . 4-151
4.89 cvround . . . 4-152
4.90 cvrvrs . . . 4-153
4.91 cvsadd . . . 4-154
4.92 cvsmaP . . . 4-155
4.93 cvsmulP . . . 4-157
4.94 cvsplit . . . 4-159
4.95 cvsqrtP . . . 4-160
4.96 cvsubP . . . 4-162
4.97 cvtrunc . . . 4-164
4.98 deq22P . . . 4-165
4.99 dolph-chebychev . . . 4-167
4.100 dotprP . . . 4-168
4.101 envelP . . . 4-169
4.102 f3x3P . . . 4-170
4.103 f5x5P . . . 4-171
4.104 fftwtsP . . . 4-172
4.105 fix2nP . . . 4-173
4.106 fix4P . . . 4-174
4.107 fix4nP . . . 4-175
4.108 fixbnP . . . 4-176
4.109 flt2P . . . 4-177
4.110 flt2iqP . . . 4-178
4.111 flt2leP . . . 4-180
4.112 flt4P . . . 4-181
4.113 flt4leP . . . 4-182
4.114 fltbP . . . 4-183
4.115 fltbuP . . . 4-184
4.116 fxsl2nP . . . 4-185
4.117 fxsl4nP . . . 4-187
4.118 fxslbnP . . . 4-189
4.119 gaussian . . . 4-191
4.120 gcexpP . . . 4-192
4.121 gcosfP . . . 4-194
4.122 hammP . . . 4-195
4.123 hannP . . . 4-196
4.124 hlbrtP . . . 4-197
4.125 imgfirP . . . 4-198
4.126 kaiser . . . 4-199
4.127 lveqP . . . 4-200
4.128 lvgeP . . . 4-202
4.129 lvgtP . . . 4-204
4.130 lvleP . . . 4-206

4.131 lvieq . . . 4-208
4.132 lvigt . . . 4-209
4.133 lvilt . . . 4-210
4.134 lvine . . . 4-211
4.135 lvinot . . . 4-212
4.136 lvltP . . . 4-213
4.137 lvneP . . . 4-215
4.138 lvnotP . . . 4-217
4.139 maxmgvP . . . 4-218
4.140 maxvP . . . 4-219
4.141 maxvi . . . 4-220
4.142 meamgvP . . . 4-221
4.143 meanc . . . 4-222
4.144 meanvP . . . 4-223
4.145 measqvP . . . 4-224
4.146 minmgvP . . . 4-225
4.147 minvP . . . 4-226
4.148 minvi . . . 4-227
4.149 mom2v . . . 4-228
4.150 mom3v . . . 4-229
4.151 mom4v . . . 4-230
4.152 mtranspose . . . 4-231
4.153 mvessqP . . . 4-232
4.154 mxcholesky . . . 4-233
4.155 mxchsolve . . . 4-234
4.156 mxmlaP . . . 4-235
4.157 mxmlsP . . . 4-237
4.158 mxmovP . . . 4-239
4.159 mxmulP . . . 4-241
4.160 mxsma . . . 4-243
4.161 mxstddev . . . 4-244
4.162 mxvariance . . . 4-245
4.163 nasCW_freewts . . . 4-246
4.164 nzcrosP . . . 4-247
4.165 polarP . . . 4-249
4.166 rectP . . . 4-251
4.167 reqsP . . . 4-253
4.168 rfftP . . . 4-254
4.169 rfftfP . . . 4-255
4.170 rfftfnP . . . 4-257
4.171 rffti . . . 4-259
4.172 rfftinP . . . 4-261
4.173 rfftscP . . . 4-263
4.174 rft2fcP . . . 4-265
4.175 rft2frP . . . 4-267
4.176 rft2ic . . . 4-269
4.177 rft2irP . . . 4-271
4.178 rfzcP . . . 4-273
4.179 rgesP . . . 4-274
4.180 rgtsP . . . 4-275
4.181 rltsP . . . 4-276
4.182 rmaxP . . . 4-277
4.183 rmaxmgP . . . 4-278
4.184 rminP . . . 4-279
4.185 rminmgP . . . 4-280
4.186 rmsqvP . . . 4-281
4.187 rnesP . . . 4-282

4.188 rsveP . . . 4-283
4.189 shphuP . . . 4-285
4.190 shphufP . . . 4-287
4.191 sn2P . . . 4-289
4.192 stddevc . . . 4-290
4.193 stddevv . . . 4-291
4.194 svdivP . . . 4-292
4.195 sveP . . . 4-293
4.196 svemgP . . . 4-294
4.197 svesqP . . . 4-295
4.198 svessqP . . . 4-296
4.199 tconvP . . . 4-297
4.200 Thread_SetParams . . . 4-299
4.201 transP . . . 4-300
4.202 vaamP . . . 4-302
4.203 vabsP . . . 4-304
4.204 vabserr . . . 4-305
4.205 vacosP . . . 4-306
4.206 vaddP . . . 4-307
4.207 vaddsub . . . 4-309
4.208 vaintP . . . 4-310
4.209 vamP . . . 4-311
4.210 vanintP . . . 4-313
4.211 variancec . . . 4-314
4.212 variancev . . . 4-315
4.213 vasbmP . . . 4-316
4.214 vasinP . . . 4-318
4.215 vasmP . . . 4-319
4.216 vatanP . . . 4-321
4.217 vatan2P . . . 4-322
4.218 vatn2fP . . . 4-324
4.219 vavexpP . . . 4-326
4.220 vavg . . . 4-327
4.221 vavgnegy . . . 4-328
4.222 vavlinP . . . 4-329
4.223 vaxyplusz . . . 4-330
4.224 vceil . . . 4-331
4.225 vclipP . . . 4-332
4.226 vcliph . . . 4-333
4.227 vclipl . . . 4-334
4.228 vclrP . . . 4-335
4.229 vcmprsP . . . 4-336
4.230 vcmpXX . . . 4-338
4.231 vcosP . . . 4-341
4.232 vcosfP . . . 4-342
4.233 vdbpwrP . . . 4-343
4.234 vdivP . . . 4-344
4.235 vdpsp . . . 4-346
4.236 veqv . . . 4-347
4.237 veucl2P . . . 4-348
4.238 veucl3P . . . 4-350
4.239 vexpP . . . 4-352
4.240 vexp2P . . . 4-353
4.241 vexp10P . . . 4-354
4.242 vexp2est . . . 4-355
4.243 vfillP . . . 4-356
4.244 vfindlastXX . . . 4-357

4.245 vfindXX . . . 4-359
4.246 vfloor . . . 4-361
4.247 vfracP . . . 4-362
4.248 vfracnP . . . 4-363
4.249 vgathrP . . . 4-364
4.250 vgenP . . . 4-365
4.251 viadd . . . 4-366
4.252 viand . . . 4-367
4.253 viars . . . 4-368
4.254 viclipP . . . 4-369
4.255 vils . . . 4-371
4.256 vimagP . . . 4-372
4.257 vifill . . . 4-374
4.258 vimov . . . 4-375
4.259 vimul . . . 4-376
4.260 vindexP . . . 4-377
4.261 vineg . . . 4-379
4.262 vintbP . . . 4-380
4.263 vintegrate . . . 4-382
4.264 vior . . . 4-383
4.265 viramp . . . 4-384
4.266 virs . . . 4-385
4.267 visdiv . . . 4-386
4.268 visequal . . . 4-387
4.269 visub . . . 4-388
4.270 viszero . . . 4-389
4.271 vixor . . . 4-390
4.272 vlimP . . . 4-391
4.273 vlintP . . . 4-393
4.274 vlmergP . . . 4-395
4.275 vlogP . . . 4-397
4.276 vlog10P . . . 4-398
4.277 vlog2P . . . 4-399
4.278 vlog2est . . . 4-400
4.279 vmaP . . . 4-401
4.280 vmaxP . . . 4-403
4.281 vmaxmgP . . . 4-405
4.282 vminP . . . 4-407
4.283 vminmgP . . . 4-409
4.284 vmmaP . . . 4-411
4.285 vmmsbP . . . 4-413
4.286 vmovP . . . 4-415
4.287 vmsaP . . . 4-416
4.288 vmsbP . . . 4-418
4.289 vmulP . . . 4-420
4.290 vnabsP . . . 4-422
4.291 vnegP . . . 4-423
4.292 vnmsaP . . . 4-424
4.293 vnorm . . . 4-426
4.294 vnormz . . . 4-427
4.295 volsave . . . 4-428
4.296 vpmergP . . . 4-429
4.297 vpolyP . . . 4-431
4.298 vpythgP . . . 4-433
4.299 vqintP . . . 4-435
4.300 vrampP . . . 4-437
4.301 vrandP . . . 4-438

4.302 vrandz . . . 4-439
4.303 vrealP . . . 4-440
4.304 vrecipP . . . 4-442
4.305 vrelerr . . . 4-443
4.306 vround . . . 4-444
4.307 vrsqrtP . . . 4-445
4.308 vrvrsP . . . 4-446
4.309 vsaddP . . . 4-447
4.310 vsaw . . . 4-448
4.311 vsbmP . . . 4-449
4.312 vsbsbmP . . . 4-451
4.313 vsbsmP . . . 4-453
4.314 vscatr . . . 4-455
4.315 vsdivP . . . 4-456
4.316 vsign . . . 4-457
4.317 vsignz . . . 4-458
4.318 vsimpsP . . . 4-459
4.319 vsinP . . . 4-460
4.320 vsinfP . . . 4-461
4.321 vsinrfP . . . 4-462
4.322 vsm2saP . . . 4-463
4.323 vsmaP . . . 4-465
4.324 vsma2P . . . 4-467
4.325 vsma2aP . . . 4-469
4.326 vsma3P . . . 4-471
4.327 vsma4P . . . 4-473
4.328 vsmsaP . . . 4-475
4.329 vsmsbP . . . 4-477
4.330 vsmulP . . . 4-478
4.331 vsort . . . 4-479
4.332 vspdp . . . 4-480
4.333 vsqP . . . 4-481
4.334 vsqrtP . . . 4-482
4.335 vsquare . . . 4-483
4.336 vssqP . . . 4-484
4.337 vsubP . . . 4-485
4.338 vsumP . . . 4-487
4.339 vswapP . . . 4-488
4.340 vtabiP . . . 4-489
4.341 vtanP . . . 4-491
4.342 vtanfP . . . 4-492
4.343 vthrP . . . 4-493
4.344 vthresP . . . 4-494
4.345 vtmergP . . . 4-495
4.346 vtrapzP . . . 4-497
4.347 vtrunc . . . 4-498
4.348 vxcsP . . . 4-499
4.349 vxcsfP . . . 4-501
4.350 wienerP . . . 4-503
4.351 zadd . . . 4-505
4.352 zconj . . . 4-506
4.353 zconv . . . 4-507
4.354 zconvd . . . 4-508
4.355 zcorr . . . 4-509
4.356 zdiv . . . 4-510
4.357 zdot . . . 4-512
4.358 zdotconj . . . 4-513

4.359 zexp . . . 4-514
4.360 zfft . . . 4-515
4.361 zfftfree . . . 4-516
4.362 zfftinit . . . 4-517
4.363 zfill . . . 4-518
4.364 zfloat16 . . . 4-519
4.365 zifft . . . 4-520
4.366 zmadd . . . 4-521
4.367 zmag . . . 4-523
4.368 zmag2 . . . 4-524
4.369 zmgetcol . . . 4-525
4.370 zmmov . . . 4-526
4.371 zmmult . . . 4-527
4.372 zmmults . . . 4-528
4.373 zmmulttr . . . 4-529
4.374 zmmulttrconj . . . 4-530
4.375 zmov . . . 4-532
4.376 zmsetcol . . . 4-533
4.377 zmsub . . . 4-534
4.378 zmtranspose . . . 4-535
4.379 zmult . . . 4-536
4.380 zmultconj . . . 4-537
4.381 zmults . . . 4-538
4.382 zneg . . . 4-539
4.383 zphase . . . 4-540
4.384 zrecip . . . 4-541
4.385 zsub . . . 4-542
4.386 zvma . . . 4-543
4.387 zvsma . . . 4-545

5 Using Routines in the Vector library 5-1
5.1 Header Files . . . 5-1
5.2 C Source Code Examples . . . 5-1
5.2.1 FFT.c, Example of Complex Numbers sing Dynamically Allocated Pointers . . . 5-2
5.2.2 FFTC.c, Example of Complex Numbers Using Two-dimensional Arrays . . . 5-5
5.2.3 FFTSC.c, Example of Complex Numbers Using Dynamically Allocated Structures . . . 5-7
5.2.4 realStride.c, Example Using stride With Real Data . . . 5-10
5.3 Makefile . . . 5-12
5.3.1 Example of Makefile for Linux . . . 5-12

6 Glossary 6-1

1 INTRODUCTION

Purpose of This Manual

This manual describes the Curtiss-Wright Vector software provided for the Curtiss-Wright Defense Solutions (CWDS) line of Intel and Power Architecture products. It is a reference containing subroutine calling sequences, descriptions and usage restrictions for all the routines available to the user. This reference also explains how to use the subroutine library (complete with example code - refer to Chapter 5, "Using Routines in the Vector Library") and lists some related terms in a glossary (refer to Section 6).

Quick Reference Guide

Section 2 of this manual gives an overview of the functionality within the Vector DSP Library. This section also acts as a quick reference guide with function names listed in a series of tables. Section 4 provides a full reference guide to each library function.

Full Reference Guide.

Section 4 of this manual provides a full reference guide to each function in the Vector DSP library. These reference pages have a common format. This format includes:

- Function name and argument declarations are given in the C language
- List of arguments, each with a one-line description and a code indicating whether it is an input, output, or both
- Description of the routine
- If applicable, restrictions and notes which must be followed to obtain the correct results

The names of each argument were chosen to provide information about the data type. The table below summarizes each argument type.

Table 1: Manual Mnemonics.

Vector Mnemonic	Scalar Mnemonic	Data Type
bv	N/A	Bytes
sv	N/A	Scalar value
iv	inc, n, ndf, mk, flg, idx	32-bit integer
rv	rs	32- or 64-bit Floating Point vector or scalar
cv	cs	Complex (32- or 64-bit Floating Point pairs)

The first letter describes the data storage format. A "**v**" as the second letter indicates that the argument is a vector or array. For example, a complex vector would be named "**cv**". A scalar argument has an "**s**" for the second letter. Integer scalars sometimes have special names to indicate what they are used for. For example: inc, n, ndf, flg, and idx. Sometimes the names end with a digit to distinguish between multiple vectors of the same type, or to make it clear how to group the arguments. "**incl**" is the increment for "**rvl**" for example.

The Vector API

The Curtiss-Wright Vector processing library supports vector and matrix operations with the following argument data types:

- Interleaved complex input and output arguments
- Split complex input and output arguments
- Real input and ouput arguments
- Integer input and output arguments

All four data types have one definition within the Vector library, apart from the Interleaved Complex data type. The Interleaved Complex data type has three definitions. The following table shows how these definitions are defined:

Table 2: Vector Data Types.

Data Type	Def No.	Definition
Split complex data:	1	two real type pointers; eg fun_name(float *cv_r, float *cv_i)
Real data:	1	one real type pointer; ie fun_name(float *rv) or fun_name(double *rv)
Integer data:	1	one integer type pointer; ie fun_name(short iv) or fun_name(int *iv) or fun_name(long *iv)
Interleaved complex data:	1	real type pointer; ie fun_name(float *cv) or fun_name(double *cv)
	2	2d real array pointer; ie fun_name(float cv[][2]) or fun_name(double cv[][2])
	3	structure pointer; ie fun_name(CV_COMPLEX *cv) or fun_name(CV_DOUBLE_COMPLEX *cv)

Which Interleaved Complex data type gets used in an application by a user depends on which set of Vector include files are defined in the application. The Vector library comes with three sets of include files. Each set of include files define a particular Interleaved Complex data type as show below:

Table 3: Vector Include Files.

Set	Interleaved Data Type	Filename	Description
1	float *cv double *cv	**CV.h** **CVD.h**	Complex/Real Single precision and Integer definitions. Complex/Real Double precision definitions;
2	float cv[][2] double *cv[][2]	**CVC.h** **SVDC.h**	Complex/Real Single precision and Integer definitions. Complex/Real Double precision definitions;
3	CV_COMPLEX *cv CV_DOUBLE_COMPLEX *cv	**CVSC.h** **CVDSC.h**	Complex/Real Single precision and Integer definitions. Complex/Real Double precision definitions;

The structures CV_COMPLEX and CV_DOUBLE_COMPLEX are defined below:

```
typedef struct
{
   float r; /*real*/
   float i; /*imaginary*/
} CV_COMPLEX;

typedef struct
{
   double r; /*real*/
   double i; /*imaginary*/
} CV_DOUBLE_COMPLEX;
```

Storage Assumptions

- Floating point values are stored according to IEEE-754 format.

Sample Increments

Most subroutines allow a sample increment to be specified for the vector argument. A sample increment of one specifies the use of every element in the vector. A sample increment of two specifies every other element. An increment of three uses every third element, etc. The very first element stored in the vector is always the first one used, regardless of the sample increment. Element skipping occurs from that point on.

A special case exists for complex vectors. To retain compatibility with existing standards, a sample increment of two is used to specify every element of a complex vector. Complex vector increases must always be even numbers. Think of the increment as counting floating point numbers. (Two floats per complex.)

Note that for all increments, the count is by elements, not by bytes. An increment of one for a short integer vector increments by two bytes (one short int), while an increment of one for a real vector increments by four bytes (one real).

Standard Restrictions

Unless otherwise noted in the description or restrictions section of the routine, the following notes apply to all Vector routines:

- An output vector must not overlap an input vector unless the operation is in-place rather than out-of-place.
- All complex vectors must have even sample increments. A sample increment of two indicates a compact complex vector, that is, no elements are skipped.
- If no sample increment argument exists for a vector, that vector is assumed to be compact.
- All arrays should be aligned as appropriate for the target hardware (See Section 3).

2 VECTOR LIBRARY OVERVIEW

The industry-standard Vector subroutine library is designed to facilitate programming and portability of signal processing code between numerous hardware platforms. Over 260 subroutines make up the current library, with functions that do FFTs, convolution, filtering, function generation and basic math. The library is based on, and built from the standards set by Floating Point Systems, Inc.; the Society of Exploration Geophysicists - Technical Standards Committee; and Quantitative Technology Corporation.

CWDS continues to add new routines to the library to round out existing sets of functions, to maintain compatibility with other vendors scientific subroutine libraries and upon request of customers. Please contact CWDS if you have a requirement not covered by the routines in the subroutine library.

The current Vector processing library is made up of both vector operations and matrix operations. These operations also include functions to perform single floating point precision, double precision, interleaved complex, split complex and integer mathematics.

The following four subsections split the Vector functions into four parts:

- Interleaved complex functions
- Split complex functions
- Real functions
- Integer functions

Interleaved Complex Functions

The following table lists the functionality available within the Vector library to carry out Interleaved Complex mathematics. Most functions are supported in Complex single and double precision. (Any function name ending with the letter **'P'** in the table supports both single and double precision)

Table 4: Interleaved Complex operations.

Function	Vector or Matrix	Description
aspecP	Vector	Accumulating Interleaved Complex Autospectrum
avcfftf	Vector	Interleaved Complex Forward FFT
avcffti	Vector	Interleaved Complex Inverse FFT
avcft2fc	Matrix	2D Interleaved Complex Forward FFT
avcft2fr	Matrix	2D Interleaved Complex Forward FFT - Row Compact
avcft2ic	Matrix	2D Interleaved Complex Inverse FFT - Column Compact
avcft2ir	Matrix	2D Interleaved Complex Inverse FFT - Row Compact
avfftwts	Vector	Create FFT Interleaved Complex and Real Exponential Tables for FFT Routines
avfreewts	Vector	Free FFT Interleaved Complex and Real Exponential Tables for FFT Routines
avrfftf	Vector	Real to Interleaved Complex Forward FFT
avrffti	Vector	Interleaved Complex to Real Inverse FFT
avrft2fc	Matrix	2D Real to Interleaved Complex Forward FFT - Column Compact
avrft2fr	Matrix	2D Real to Interleaved Complex Forward FFT - Row Compact
avrft2ic	Matrix	2D Real to Interleaved Complex Inverse FFT - Column Compact
avrft2ir	Matrix	2D Real to Interleaved Complex Inverse FFT - Row Compact
avtcft2fc	Matrix	Multi-threaded 2D Interleaved Complex Fwd FFT - Column Compact
avtcft2fr	Matrix	Multi-threaded 2D Interleaved Complex Fwd FFT - Row Compact
avtcft2ic	Matrix	Multi-threaded 2D Interleaved Complex Inv FFT - Column Compact
avtcft2ir	Matrix	Multi-threaded 2D Interleaved Complex Inv FFT - Row Compact
avtrft2fc	Matrix	Multi-threaded 2D Real to Interleaved Complex Fwd FFT - Column Compact
avtrft2fr	Matrix	Multi-threaded 2D Real to Interleaved Complex Fwd FFT - Row Compact
avtrft2ic	Matrix	Multi-threaded 2D Interleaved Complex to Real Inv FFT - Column Compact
avtrft2ir	Matrix	Multi-threaded 2D Interleaved Complex to Real Inv FFT - Row Compact
ccdotpP	Vector	Interleaved Complex Conjugate Dot Product
cconvdP	Vector	Interleaved Complex Convolution with Decimation
cdotprP	Vector	Interleaved Complex Dot Product
cfftP	Vector	Interleaved Complex FFT, in Place - Fwd and Inv
cfftfP	Vector	Interleaved Complex Forward FFT
cfftfnP	Vector	Interleaved Complex Fwd FFT (Non-Destructive)
cfftiP	Vector	Interleaved Complex Inverse FFT
cfftinP	Vector	Interleaved Complex Inv FFT (Non-Destructive)
cfftscP	Vector	Interleaved Complex FFT Scale
cft2fcP	Matrix	2D Interleaved Complex Forward FFT - Column Compact
cft2frP	Matrix	2D Interleaved Complex Forward FFT - Row Compact
cft2icP	Matrix	2D Interleaved Complex Inverse FFT - Column Compact
cft2irP	Matrix	2D Interleaved Complex Inverse FFT - Row Compact
cpows	Vector	Interleaved Complex Vector Power with Add
crvdivP	Vector	Interleaved Complex Vector Real Vector Divide
crvmulP	Vector	Interleaved Complex Vector Real Vector Multiply
cspecP	Vector	Accumulating Interleaved Complex Cross Spectrum
csve	Vector	Interleaved Complex Sum Vector Elements
cvabsP	Vector	Interleaved Complex Vector Absolute Value
cvabserr	Vector	Interleaved Complex Absolute Error
cvaddP	Vector	Interleaved Complex Vector Add
cvaddsub	Vector	Interleaved Complex Add/Subtract
cvavg	Vector	Interleaved Complex Average

cvavgnegy	Vector	Interleaved Complex Average x, -y
cvaxyplusz	Vector	Interleaved Complex aXY + Z
cvceil	Vector	Interleaved Complex Ceiling
cvclip	Vector	Interleaved Complex Clip
cvcliph	Vector	Interleaved Complex Clip High
cvclipl	Vector	Interleaved Complex Clip Low
cvcmlP	Vector	Interleaved Complex Vector Conjugate Multiply
cvcmlaP	Vector	Interleaved Complex Vector Conjugate Multiply with Add
cvcmpXX	Vector	Compare Interleaved Complex Vectors
cvcombP	Vector	Form Interleaved Complex Vector from Two Real Vectors
cvconjP	Vector	Interleaved Complex Vector Conjugate
cvcsmlP	Vector	Interleaved Complex Vector Complex Scalar Multiply
cvcvrsmlP	Vector	Interleaved Complex Vector Complex Vector Real Scalar Multiply
cvdivP	Vector	Interleaved Complex Vector Complex Vector Division
cvexpP	Vector	Interleaved Complex Vector Exponential
cvfillP	Vector	Set a Interleaved Complex Vector to a Complex Scalar
cvfindlastXX	Vector	Find Element in a Interleaved Complex Vector
cvfindXX	Vector	Find Element in a Interleaved Complex Vector
cvfloor	Vector	Interleaved Complex Floor
cvintegrate	Vector	Discrete Interleaved Complex Integrator
cvmaP	Vector	Interleaved Complex or Conjugate Vector Multiply with Add
cvmagsP	Vector	Interleaved Complex Vector Magnitude Squared
cvmerge	Vector	Merge Real, Imag into Interleaved Complex
cvmexpP	Vector	Interleaved Complex Vector Exponential with Multiply
cvmgsaP	Vector	Interleaved Complex Vector Magnitude Squared with Add
cvmlP	Vector	Interleaved Complex Vector Multiply
cvmlaP	Vector	Interleaved Complex Vector Multiply with Add
cvmovP	Vector	Interleaved Complex Vector Move
cvmulP	Vector	Interleaved Complex Vector Multiply or Conjugate Multiply
cvnegP	Vector	Interleaved Complex Vector Negate
cvolsave	Vector	Interleaved Complex Overlap and Save
cvphasP	Vector	Interleaved Complex Vector Phase
cvrcipP	Vector	Interleaved Complex Vector Reciprocal
cvrealP	Vector	Make a Interleaved Complex Vector from a Real Vector
cvrelerr	Vector	Interleaved Complex Relative Error
cvround	Vector	Interleaved Complex Round
cvrvrs	Vector	Interleaved Complex Reverse
cvsadd	Vector	Interleaved Complex Scalar Add
cvsmaP	Vector	Scale Interleaved Complex Vector and Add to Second Complex Vector
cvsmulP	Vector	Interleaved Complex Vector Scalar Multiply
cvsplit	Vector	Interleaved Complex into Real, Imag
cvsqrtP	Vector	Interleaved Complex Vector Square Root.
cvsubP	Vector	Interleaved Complex Vector Subtract
cvtrunc	Vector	Interleaved Complex Truncate
fftwtsP	Vector	Create FFT Interleaved Complex Exponential Tables
flt2iqP	Vector	Float Integer (2 Byte) I,Q pairs and demux 2 channels
gcexpP	Vector	Generate a Interleaved Complex Exponential with Constant Rotation
meanc	Vector	Interleaved Complex Mean of Vector Elements
stddevc	Vector	Standard Deviation of Interleaved Complex Vector
transP	Vector	Interleaved Complex Vector Divided by Real Vector (Transfer)
variancec	Vector	Variance of Interleaved Complex Vector
vimagP	Vector	Extract Imaginary Part of Interleaved Complex Vector
vrealP	Vector	Extract Real Part of Interleaved Complex Vector
vxcsP	Vector	Real Vector Multiplied by Interleaved Complex Exponential
vxcsfP	Vector	Real Vector Multiplied by Fractional Complex Exponential
polarP	Vector	Rectangular to Polar Coordinate Conversion
rectP	Vector	Polar to Rectangular Coordinate Conversion

rfftP	Vector	Real FFT, in Place - Fwd and Inv
rfftfP	Vector	Real to Interleaved Complex Forward FFT
rfftfnP	Vector	Real to Interleaved Complex Forward FFT (Non-Destructive)
rffti	Vector	Interleaved Complex to Real Inverse FFT
rfftinP	Vector	Interleaved Complex to Real Inverse FFT (Non-Destructive)
rfftscP	Vector	Real FFT Scale and Format
rft2fcP	Matrix	2D Real to Interleaved Complex Forward FFT - Column Compact
rft2frP	Matrix	2D Real to Interleaved Complex Forward FFT - Row Compact
rft2ic	Matrix	2D Interleaved Complex to Real Inverse FFT - Column Compact
rft2irP	Matrix	2D Interleaved Complex to Real Inverse FFT - Row Compact

Complex Split Functions

The z series of functions (zmult, etc.) operate on complex split data. Complex split data represents a complex vector using two separate vectors, one containing real data, the other containing imaginary data. This is in contrast to complex interleaved data, where each complex sample is represented by a pair of floating point values (real, imaginary) that are in adjacent memory locations.

Many operations on complex split data are faster than operations on interleaved data. This is a consequence of machine instructions. Also, complex split data has the advantage that the separate real or imaginary vectors can be processed independently, if desired, using real vector functions.

Table 5: Split Complex operations.

Function	Vector or Matrix	Description
zadd	Vector	Split Complex Add
zconj	Vector	Split Complex Conjugate
zconv	Vector	Split Complex Convolution
zconvd	Vector	Split Complex Decimating Convolution
zcorr	Vector	Split Complex Correlation
zdiv	Vector	Split Complex Divide
zdot	Vector	Split Complex Dot Product
zdotconj	Vector	Split Complex Dot Product Conjugate
zexp	Vector	Split Complex Exponentiate
zfft	Vector	Split Complex FFT
zfftfree	Vector	Split Complex Deallocate FFT Weights
zfftinit	Vector	Split Complex FFT Weights
zfill	Vector	Split Complex Fill
zfloat16	Vector	Split Complex Float
zifft	Vector	Split Complex Inverse FFT
zmadd	Matrix	Split Complex Matrix Add
zmag	Vector	Magnitude of Split Complex Vector
zmag2	Vector	Magnitude Squared of Split Complex Vector
zmgetcol	Matrix	Split Complex Get a Matrix Column
zmmov	Matrix	Split Complex Sub-Matrix Move
zmmult	Matrix	Split Complex Matrix Multiply
zmmultconj	Matrix	Split Complex Matrix Multiply Conjugate
zmmults	Matrix	Split Complex Matrix Multiply by Scalar
zmmulttr	Matrix	Split Complex Matrix Multiply Transpose
zmmulttrconj	Matrix	Split Complex Matrix Multiply Transpose Conjugate
zmov	Vector	Split Complex Move
zmsetcol	Matrix	Split Complex Set a Matrix Column
zmsub	Matrix	Split Complex Matrix Subtract
zmtranspose	Matrix	Split Complex Matrix Transpose

zmult	Vector	Split Complex Multiply
zmultconj	Vector	Split Complex Multiply Conjugate
zmults	Vector	Split Complex Multiply Scalar
zneg	Vector	Split Complex Negate
zphase	Vector	Phase of Split Complex Vector
zrecip	Vector	Split Complex Reciprocal
zsub	Vector	Split Complex Subtract
zvma	Vector	Split Complex Multiply/Add
zvsma	Vector	Split Complex Scalar Multiply/Add

The above split complex fuctions are only implemented in single floating point precision. At this present time Split complex in double precision is not supported.

Although the z functions operate on complex split data, many functions also accept real data operands as well. So two call to these functions can process the complex split data if the real and imaginary parts can be processed independantly.

Real Vector Functions.

The following list of functions operate with real vectors and matrices. Most of the listed functions have both single and double precision versions. (Any function name ending with the letter **'P'** in the table supports both single and double precision)

Table 6: Real data operations.

Function	Vector or Matrix	Description
acorfP	Vector	Frequency Domain Auto Correlation
acortP	Vector	Time Domain Auto Correlation
blkmanP	Vector	Apply a Blackman Window to a Real Vector
blkman-harris	Vector	Blackman-Harris Window
convdP	Vector	Convolution with Decimation
deq22P	Vector	Difference Equation, 2 Poles, 2 Zeroes
dolph-chebychev	Vector	Dolph-Chebychev Window
dotprP	Vector	Real Dot Product
envelP	Vector	Vector Envelope
f3x3P	Matrix	Filter 3x3 Convolution
f5x5P	Matrix	Filter 5x5 Convolution
fix2nP	Vector	Vector Fix to Two-byte Integer and Round
fix4P	Vector	Vector Fix to Four-byte Integer and Truncate
fix4nP	Vector	Vector Fix to Four-byte Integer and Round
fixbnP	Vector	Vector Fix to One-byte Integer and Round
flt2P	Vector	Float Integer (2 Byte) Vector
flt2leP	Vector	Float Integer (2 Byte Little Endian) Vector
flt4P	Vector	Float Integer (4 Byte) Vector
flt4leP	Vector	Float Integer (4 Byte Little Endian) Vector
fltbP	Vector	Float Signed Byte
fltbuP	Vector	Float Byte Unsigned
fxsl2nP	Vector	Vector Scale, Limit, Fix to Two-byte Integer and Round
fxsl4nP	Vector	Vector Scale, Limit, Fix to Four-byte Integer and Round
fxslbnP	Vector	Vector Scale, Limit, Fix to One-byte Integer and Round
gaussian	Vector	Gaussian Window
gcosfP	Vector	Vector Generate Cosines
hammP	Vector	Apply a Hamming Window to a Real Vector
hannP	Vector	Apply a Hanning Window to a Real Vector
hlbrtP	Vector	Hilbert Transform
imgfirP	Vector	Filter MxN Convolution

kaiser	Vector	Kaiser Window
lveqP	Vector	Logical Vector Equal
lvgeP	Vector	Logical Vector Greater Than or Equal
lvgtP	Vector	Logical Vector Greater
lvleP	Vector	Logical Vector Less Than or Equal
lvltP	Vector	Logical Vector Less
lvneP	Vector	Logical Vector Not Equal
lvnotP	Vector	Logical Vector Not Equal to Zero
maxmgvP	Vector	Maximum Magnitude Element of Vector
maxvP	Vector	Find the Maximum Value and its Location
meamgvP	Vector	Mean Magnitude of Real Vector
meanvP	Vector	Mean of Real Vector
measqvP	Vector	Mean of Square of Real Vector
minmgvP	Vector	Find the Minimum Magnitude and its Location
minvP	Vector	Find the Minimum Value and its Position
mom2v	Vector	2nd Moment of a Vector
mom3v	Vector	3rd Moment of a Vector
mom4v	Vector	4th Moment of a Vector
mtranspose	Matrix	Matrix Transpose
mvessqP	Vector	Mean of Signed Square of Vector Elements
mxcholesky	Matrix	Cholesky Decomposition
mxchsolve	Matrix	Cholesky Solve
mxmlaP	Matrix	Real Matrix Multiply and Add
mxmlsP	Matrix	Real Matrix Multiply and Subtract
mxmovP	Matrix	Real Matrix / Submatrix move
mxmulP	Matrix	Real Matrix Multiply
mxsma	Matrix	Real Scalar Matrix Multiply and Add
mxstddev	Matrix	Standard Deviation of Real Matrix
mxvariance	Matrix	Variance of Real Matrix
nzcrosP	Vector	Find Location of Selected Zero Crossing
reqsP	Vector	Find Location of First Element Equal to a Scalar
rfzcP	Vector	Find Location of First Zero Crossing
rgesP	Vector	Find Location of First Element Greater/Equal to a Scalar
rgtsP	Vector	Find Location of First Element Greater Than a Scalar
rltsP	Vector	Find Location of First Element Less Than a Scalar
rmaxP	Vector	Find the Maximum Value and its Location
rmaxmgP	Vector	Find the Maximum Magnitude
rminP	Vector	Find the Minimum Value and its Position
rminmgP	Vector	Find the Minimum Magnitude
rmsqvP	Vector	Root Mean Square of a Real Vector
rnesP	Vector	Find Position of First Element Not Equal to a Scalar
rsveP	Vector	Running Sum of Real Vector
shphuP	Vector	SchaferâĂŹs Phase Unwrapping
shphufP	Vector	SchaferâĂŹs Phase Unwrapping, Fraction of a Circle Argument
sn2P	Vector	Sum the Squared Difference Between Two Vectors
stddevv	Vector	Standard Deviation of Real Vector
svdivP	Vector	Divide Scalar by Vector
sveP	Vector	Sum of Real Vector
svemgP	Vector	Sum of Vector Magnitudes
svesqP	Vector	Sum of Vector Elements Squared
svessqP	Vector	Sum of Vector Element Signed Squares
vaamP	Vector	Vector Add, Add, and Multiply
vabsP	Vector	Vector Absolute Value
vabserr	Vector	Absolute Error
vacosP	Vector	Vector Arccosine.
vaddP	Vector	Add Two Vectors.
vaddsub	Vector	Real Add/Subtract

vaintP	Vector	Vector Align to Integer
vamP	Vector	Vector Add and Multiply
vanintP	Vector	Vector Align to Nearest Integer
variancev	Vector	Variance of Real Vector
vasbmP	Vector	Vector Add, Subtract, and Multiply
vasinP	Vector	Vector Arcsine.
vasmP	Vector	Vector Add and Scalar Multiply
vatanP	Vector	Vector Arctangent.
vatan2P	Vector	Vector Arctangent of Two Arguments
vatn2fP	Vector	Vector Arctangent of Two Arguments in Fractions
vavexpP	Vector	Vector Exponential Averaging
vavg	Vector	Average
vavgnegy	Vector	Average x, -y
vavlinP	Vector	Vector Linear Averaging
vavlinP	Vector	Vector Linear Averaging
vaxyplusz	Vector	Real aXY + Z
vceil	Vector	Ceiling
vclipP	Vector	Vector Clip
vcliph	Vector	Clip High
vclipl	Vector	Clip Low
vclrP	Vector	Zero a Vector
vcmprsP	Vector	Vector Compress
vcmpXX	Vector	Compare Vectors
vcosP	Vector	Vector Cosine
vcosfP	Vector	Vector Cosine in Fractions
vdbpwrP	Vector	Vector Conversion to dB
vdivP	Vector	Divide One Vector by Another
vdpsp	Vector	Vector Convert Double to Single Precision
veucl2P	Vector	Vector Euclidean Distance
veucl3P	Vector	Vector Euclidean Distance (3 Dimensional)
vexpP	Vector	Vector Exponentiation
vexp2P	Vector	Vector Base 2 Exponential
vexp10P	Vector	Vector Base 10 Exponential
vexp2est	Vector	exp2 Estimate
vfillP	Vector	Set a Vector in Memory to a Scalar Value
vfindlastXX	Vector	Find Element in a Vector
vfindXX	Vector	Find Element in a Vector
vfloor	Vector	Floor
vfracP	Vector	Vector Truncate to Fraction
vfracnP	Vector	Vector Truncate to Nearest Fraction
vgathrP	Vector	Vector Gather
vgenP	Vector	Generate a Vector in Memory
viclipP	Vector	Vector Inverse Clip
vindexP	Vector	Vector Index, Truncate
vintbP	Vector	Vector Interpolate
vintegrate	Vector	Discrete Integrator
visequal	Vector	Test Vectors for Equality
vlimP	Vector	Vector Limit
vlintP	Vector	Vector Linear Interpolate
vlmergP	Vector	Vector Logical Merge
vlogP	Vector	Compute the Natural Logarithm of a Vector
vlog10P	Vector	Vector Base 10 Logarithm
vlog2P	Vector	Vector Base 2 Logarithm
vlog2est	Vector	Log2 Estimate
vmaP	Vector	Vector Multiply and Add
vmaxP	Vector	Vector Maximum
vmaxmgP	Vector	Vector Maximum Magnitude

vminP	Vector	Vector Minimum
vminmgP	Vector	Vector Minimum Magnitude
vmmaP	Vector	Vector Multiply, Multiply, and Add
vmmsbP	Vector	Vector Multiply, Multiply, and Subtract
vmovP	Vector	Copy One Vector to Another
vmsaP	Vector	Vector Multiply and Scalar Add
vmsbP	Vector	Vector Multiply and Subtract
vmulP	Vector	Multiply Two Vectors
vnabsP	Vector	Vector Negative Absolute Value
vnegP	Vector	Negate a Vector
vnmsaP	Vector	Vector Negative Multiply and Scalar Add
vnorm	Vector	Generate a normally-distributed vector
volsave	Vector	Overlap and Save
vpmergP	Vector	Vector Positive Merge
vpolyP	Vector	Vector Polynomial Evaluation
vpythgP	Vector	Vector Pythagorean
vqintP	Vector	Vector Quadratic Interpolate
vrampP	Vector	Generate a Ramp in a Vector
vrandP	Vector	Random Number Generator
vrecipP	Vector	Compute Reciprocal of Vector
vrelerr	Vector	Relative Error
vround	Vector	Round
vrsqrtP	Vector	Vector Reciprocal Square Root
vrvrsP	Vector	Reverse a Vector
vsaddP	Vector	Add a Scalar to a Vector
vsaw	Vector	Generate Sawtooth Signal
vsbmP	Vector	Vector Subtract and Multiply
vsbsbmP	Vector	Vector Subtract, Subtract, and Multiply
vsbsmP	Vector	Vector Subtract and Scalar Multiply
vscatr	Vector	Vector Scatter
vsdivP	Vector	Divide Vector by Scalar
vsign	Vector	Signum Function
vsignz	Vector	Signum Function, Negative 0
vsimpsP	Vector	SimpsonâĂŹs Rule Integration
vsinP	Vector	Vector Sine.
vsinfP	Vector	Vector Sine in Fractions.
vsinrfP	Vector	Vector Sine in Fractions, Reduced Range.
vsm2saP	Vector	Multiply Two Vectors by Scalars and Add a Scalar
vsmaP	Vector	Vector Scalar Multiply and Add
vsma2P	Vector	Two Vector Multiply and Scalar Add
vsma2aP	Vector	Two Vector Multiply and Scalar Add, Accumulate
vsma3P	Vector	Three Vector Multiply and Scalar Add
vsma4P	Vector	Four Vector Multiply and Scalar Add
vsmsaP	Vector	Multiply Vector by a Scalar and Add Scalar
vsmsbP	Vector	Vector Scalar Multiply and Subtract
vsmulP	Vector	Multiply Vector by a Scalar
vsort	Vector	Vector Sort
vspdp	Vector	Vector Convert Single to Double Precision
vsqP	Vector	Vector Square
vsqrtP	Vector	Vector Square Root
vsquare	Vector	Generate Square Wave Signal
vssqP	Vector	Vector Signed Square
vsubP	Vector	Subtract One Vector from Another
vsumP	Vector	Vector Sum
vswapP	Vector	Vector Swap
vtabiP	Vector	Vector Table Look-up, Linear Interpolate
vtanP	Vector	Vector Tangent

vtanfP	Vector	Vector Tangent in Fractions of a Circle
vthresP	Vector	Replace Elements Less Than Scalar with Zero
vtmergP	Vector	Vector Tapered Merge
vtrapzP	Vector	Trapezoidal Rule Integration
vtrunc	Vector	Truncate
wienerP	Vector	Wiener Levinson Equation Solver

Integer Vector Functions.

The following table shows the integer support currently available within the Vector Library:

Table 7: Integer operations.

Function	Vector or Matrix	Description
fix2nP	Vector	Vector Fix to Two-byte Integer and Round
fix4P	Vector	Vector Fix to Four-byte Integer and Truncate
fix4nP	Vector	Vector Fix to Four-byte Integer and Round
fixbnP	Vector	Vector Fix to One-byte Integer and Round
flt2P	Vector	Float Integer (2 Byte) Vector
flt2iqP	Vector	Float Integer (2 Byte) I,Q pairs and demux 2 channels
flt2leP	Vector	Float Integer (2 Byte Little Endian) Vector
flt4P	Vector	Float Integer (4 Byte) Vector
flt4leP	Vector	Float Integer (4 Byte Little Endian) Vector
fxsl2nP	Vector	Vector Scale, Limit, Fix to Two-byte Integer and Round
fxsl4nP	Vector	Vector Scale, Limit, Fix to Four-byte Integer and Round
fxslbnP	Vector	Vector Scale, Limit, Fix to One-byte Integer and Round
lvieq	Vector	Integer Vector Equal
lvigt	Vector	Integer Vector Greater
lvilt	Vector	Integer Vector Less Than
lvine	Vector	Integer Vector Not Equal
lvinot	Vector	Integer Complement
maxvi	Vector	Integer Max Index
minvi	Vector	Integer Min Index
vaintP	Vector	Vector Align to Integer
veqv	Vector	Exclusive NOR (XNOR) Two Integer Vectors
viadd	Vector	Add Two Integer Vectors
viand	Vector	And Two Integer Vectors
viars	Vector	Vector Integer Arithmetic Right Shift
viclipP	Vector	Vector Inverse Clip
vils	Vector	Vector Integer Left Shift
vifill	Vector	Integer Vector Fill
vimov	Vector	Integer Vector Move
vimul	Vector	Multiply Two Integer Vectors
vineg	Vector	Vector Integer Negate
vior	Vector	Vector Integer OR Two Integer Vectors
viramp	Vector	Integer Ramp Generate
virs	Vector	Vector Integer Right Shift
visdiv	Vector	Integer Vector Scalar Divide
visub	Vector	Subtract Two Integer Vectors
vixor	Vector	Exclusive OR (XOR) Two Integer Vectors

There is currently no matrix support within the integer operations.

3 GETTING THE BEST PERFORMANCE

This section is a short guide for programmers using the Vector Library. It contains explanations of library behaviour, and tips on selecting the right storage options for your data to increase performance.

Memory Alignment

The efficiency of many operations is improved if data within memory is correctly aligned on certain word boundaries. Vectors and matrices can be loaded and stored faster if they are vector aligned. The following table gives the vector alignment and minimum vector length for float data:

Technology	Vector Aligment
SSE	16
AVX	32
AltiVec	16

Alignment can be controlled using a function such as `memalign`. This is a C function that is not in the ANSI standard but is available on many systems. It is defined in `malloc.h` on Linux systems. The following macro redefines `malloc` so that all memory allocation is optimally aligned:

```
#include <malloc.h>
#define malloc(SIZE) memalign(16, SIZE)
```

Some operating systems (*e.g.* Apple's OSX) automatically align all memory to a 16-byte boundary so `memalign` is not needed.

Vector/Matrix Format

When available, vector and matrix calculations are done using single instruction, multiple data (SIMD) instructions to process several elements simultaneously. This imposes a minimum vector length given in the table below:

Technology Vector Length (floats)	minimum
SSE	4
AVX	8
AltiVec	4

For short ints (16 bits) the vector length should be twice that of the float vector length. If the vector unit supports doubles (64 bits), then the vector length should be half that of floats.
For best performance all input and output vectors should:

- have a stride of 1

  Vectors and matrices can be loaded and stored much quicker when they are contiguous in memory. The library includes special optimisations for a stride length of 2 (which was added for interleaved complex numbers), but all other non-unit strides will be significantly slower than a stride of 1 and, in many cases, almost as slow as unvectorised scalar code. Note that, a stride of -1 will also be significantly slower than a stride of $+1$.

- be vector aligned

- have length greater than or equal to the vector length

  The vector unit works on arrays of the vector length so no speed up is gained by using the library on vectors of length less than this.

- have row (row major matrices) or column (column major matrices) length divisible by the vector length

  For a row major matrix: if the row length of a matrix is not divisible by the vector length then the alignment of the first element of each row will change for each row/column. For optimal performance the first element of each row should be vector aligned.

  The same rule applies to columns in column major matrices.

- have a length divisible by the vector length

  Any elements at the end of the vector which cannot be dealt with by the vector unit must be dealt with in normal scalar code, which will decrease the performance. The decrease in performance becomes less important for longer vectors.

Error Checking and Debugging

Two versions of the Vector library are provided: a performance version and a development version. The development version of the library (signified by a 'D' in the library's name) contains full error checking and should always be used when developing and debugging applications.
A few library functions return status information: always check the return code of those that do.
The performance version of the library contains no error checking, and consequently runs faster than the development library. The performance library should only be used with applications that have been run successfully with the development version of the library.
When timing code, the performance version of the library should be used.
Note: the performance version of the library reads in data before it knows how much will be used and as a result often reads more data than is needed. This is not a problem, except when using memory checkers such as Electric Fence which object to this behaviour. The development library only reads in the data it intends to use and so is safe to use with memory checkers.

Scalar Functions

As the vector unit works on arrays of the vector length, scalar functions in the library are not vectorised.

Random Number Generation

The random number generation functions have not been vectorised in the current version of the library.

Vector and Elementwise Operations

All vector and elementwise operations work optimally on vectors which match the conditions given in section 3.

Signal Processing Functions

All signal processing operations work optimally on vectors which match the conditions given in section 3.

FFT Functions

To get the best performance from an FFT, a vector must have length a multiple of the numbers 2, 4, 8, and 3 only. If a vector length is not a multiple of these numbers, a DFT may be done, which is considerably slower than an FFT.

Factors of 3 should be avoided if possible. An FFT will only be done for factors of 3 if the length also has a factor of 16, otherwise a DFT is done.

When doing large FFT's, optimal routines have been developed for the lengths: 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, and 65536. These lengths should be much quicker than lengths of similar magnitude.

In-place FFT's are normally faster than out-of-place FFT's.

FFT's are fastest with a scale factor of 1. However, if you need to use a different scale factor, it is better to let the FFT routine do the scaling rather than to do it yourself.

The internal FFT routines only work on vector aligned data with a stride of 1. If vectors are used which do not match these restrictions an internal copy of the vector will be made. This is an important consideration when using large vectors. Also, if complex vectors are not stored split an internal copy will be made.

FIR Filter, Convolution and Correlation Functions

These functions call the FFT functions internally and are therefore subject to the same restrictions.

Linear Algebra Functions

For optimal performance the vectors and matrices used with the linear algebra functions should match the conditions given in Section 3.

Matrix and Vector Operations

Matrix and vector operations should work optimally on row or column major matrices (row major is the default), however, the restriction exists that all matrices passed to a function should be of the same order. For example, using two row major matrices as input to a function and a column major as output will be slower than using all row major or all column major. When matrices are passed to Vector functions that are not all of the same order, the library will assume they are all row major and treat the column major matrices as strided matrices.

Controlling the Number of Threads

The Threaded Vector library is multithreaded and will take advantage of multiple cores on the processor invoking it. Utilising multiple threads is automatic:

- The maximum number of threads used is set when the Vector initialization function is called.
- The maximum number running at any one time is also set at that point. If a threaded routine is called with (say) 4 threads and we have hit this maximum number running then four of them are shut down before the new function is executed.
- The number of threads invoked when a routine is called, is decided by that routine by reference to the data (vector or matrix) size specified in the call, to provide the best performance for that call.

It is possible to change the maximum number of threads used.

1. A threaded, and a non-threaded ("serial") version of the library are provided. If you wish to only ever use one thread in a library call, use the serial version of the library.
2. The maximum number of threads used for a specific function call, and the maximum number kept running at any one time, can be changed by a call to the routine `Thread_SetParams` with arguments `num_threads` and `max_num_running`. This call, if used, *must* be made before the Vector initialization function is called.
3. When calling the routine `Thread_SetParams` the value of `max_num_running` must be greater or equal to 3*`num_threads`. If the user enters a smaller value than this in their `Thread_SetParams` function call then the function will set the value of `max_num_running` to 3*`num_threads`.
4. If no call to `Thread_SetParams` is made, the library default values will be utilised.
5. The function `Thread_SetParams` is now defined in the Vector library include files. Therefore, it may be called without defining it as an external function in the application code.

Support Functions

Always call `nas_init` and `nas_finalize` at the beginning and end of a program. The finalize function will free up any used memory at the end of the program by resources such as the thread management system. Failure to call the finalize function will result in a small amount of memory leakage.

4 Vector Library Function Reference

acorfP

Frequency Domain Auto Correlation

Function Definitions

```
void acorf(
                    float *rv1,
                    float *rvo,
                    long   n0,
                    long   n1);

void acorf_(
                    float *rv1,
                    float *rvo,
                    long  *n0,
                    long  *n1);

void acorfd(
                    double *rv1,
                    double *rvo,
                    long    n0,
                    long    n1);

void acorfd_(
                    double *rv1,
                    double *rvo,
                    long   *n0,
                    long   *n1);
```

Parameters

Argument	I/O	Description
`rv1`	input	float or double vector.
`rvo`	output	float or double vector.
`n0`	input	long integer scalar (points in rvo).
`n1`	input	long integer scalar (points in rv1).

Return Value

None: void function.

Description

Auto correlates in the frequency domain the input vector, rv1, with a size of n1, producing the output vector, rvo, with a size no.

Restrictions and Notes

1. The storage allocated by rv1 must be twice the size of n1.
2. All vectors must be compact.
3. This is an out-of-place operation only.
4. The operation is performed faster if the input and output buffers are fully aligned for the target hardware. This operation is supported in both single and double precision.
5. This operation is supported in both single and double precision.

acortP

Time Domain Auto Correlation

Function Definitions

```
void acort(
                    float *rv1,
                    float *rvo,
                    long   n0,
                    long   n1);

void acort_(
                    float *rv1,
                    float *rvo,
                    long  *n0,
                    long  *n1);

void acortd(
                    double *rv1,
                    double *rvo,
                    long    n0,
                    long    n1);

void acortd_(
                    double *rv1,
                    double *rvo,
                    long   *n0,
                    long   *n1);
```

Parameters

Argument	I/O	Description
rv1	input	float or double vector.
rvo	output	float or double vector.
n0	input	long integer scalar (points in rvo).
n1	input	long integer scalar (points in rv1).

Return Value

None: void function.

Description

Auto correlates in the time domain the input vector, rv1, with a size of n1, producing the output vector, rvo, with a size of no.

Restrictions and Notes

1. All vectors must be compact.
2. This is an out-of-place operation only.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

aspecP

Accumulating Autospectrum

Function Definitions

```
void aspec(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   inc0,
                    long   n);

void aspec_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *inc0,
                    long  *n);

void aspecd(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   inc0,
                    long   n);

void aspecd_(
                    doube  *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *inc0,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double vector.
cv1inc	input	long integer scalar (cv1 vector stride).
cvo	output	complex float or double vector.
cvoinc	input	long integer scalar (cvo vector stride).
n	input	long integer scalar (sample length).

Return Value

None: void function.

Description

Each element of the complex input vector, cv1, is multiplied by its conjugate and added to the real input vector, rvio. The result is placed back in the real vector, rvio, becoming the real output vector. (This routine is included only for backwards compatibility with other subroutine libraries. For new programs, use cvmgsa.)

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. This is an out-of-place operation only.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

avcfftf

Complex to Complex Forward FFT

Function Definitions

```
void avcfftf(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n,
                    long  *p);

void avcfftf_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double vector.
cv1inc	input	long integer scalar (cv1 vector stride).
cvo	output	complex float or double vector.
cvoinc	input	long integer scalar (cvo vector stride).
n	input	long integer scalar (sample length).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The forward complex FFT of the complex input vector is computed and stored in the complex output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avcffti

Complex to Complex Inverse FFT

Function Definitions

```
void avcffti(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n,
                    long  *p);

void avcffti_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double vector.
cv1inc	input	long integer scalar (cv1 vector stride).
cvo	output	complex float or double vector.
cvoinc	input	long integer scalar (cvo vector stride).
n	input	long integer scalar (sample length).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The inverse complex FFT of the complex input vector is computed and stored in the complex output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avcft2fc

2D Complex to Complex Forward FFT

Function Definitions

```
void avcft2fc(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p);

void avcft2fc_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double matrix.
cvo	output	complex float or double matrix.
nr	input	long integer scalar (No of rows).
nc	input	long integer scalar (No of cols).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input matrix is computed and stored in the output matrix. The columns are assumed to be compact. The two dimensional forward FFT is performed as a series of forward complex-to-complex FFTs across the rows, followed by complex-to-complex forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avcft2fr

2D Complex to Complex Forward FFT - Row Compact

Function Definitions

```
void avcft2fr(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p);

void avcft2fr_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double matrix.
cvo	output	complex float or double matrix.
nr	input	long integer scalar (No of rows).
nc	input	long integer scalar (No of cols).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional forward FFT is performed as a series of forward complex-to-complex FFTs across the rows, followed by complex-to-complex forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avcft2ic

2D Complex to Complex Inverse FFT - Column Compact

Function Definitions

```
void avcft2ic(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p);

void avcft2ic_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double matrix.
cvo	output	complex float or double matrix.
nr	input	long integer scalar (No of rows).
nc	input	long integer scalar (No of cols).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The two dimensional inverse complex FFT of the input matrix is computed and stored in the output matrix. The columns are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs down the columns, followed by complex-to-complex inverse FFTs across the rows.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avcft2ir

2D Complex to Complex Inverse FFT - Row Compact

Function Definitions

```
void avcft2ir(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p);

void avcft2ir_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double matrix.
cvo	output	complex float or double matrix.
nr	input	long integer scalar (No of rows).
nc	input	long integer scalar (No of cols).
p	input	pointer to FFT setup generated by avfftwts.

Return Value

None: void function.

Description

The two dimensional inverse complex FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs down the columns, followed by complex-to-complex inverse FFTs across the row

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avfftwts

Create FFT Complex and Real Exponential Tables for FFT Routines

Function Definitions

```
void avfftwts(
                    long *p,
                    long  m,
                    long  n);

void avfftwts_(
                    long *p,
                    long *m,
                    long *n);
```

Parameters

Argument	I/O	Description
p	output	Pointer to a structure containing FFT setup.
m	input	Max Complex FFT size (number of complex points).
n	input	Real FFT size (number of real points).

Return Value

None: void function.

Description

A lookup table of complex numbers is computed for performing FFTs consisting of up to m complex samples. Optionally a second lookup table of complex numbers is computed for performing FFTs consisting of n real samples. A structure is created containing the sizes and base addresses of the generated lookup table(s). The address of that structure is returned as the value of p.

Restrictions and Notes

1. The FFTs are performed faster if the FFT lengths are a power of two.

avfreewts

Free FFT Complex and Real Exponential Tables for FFT Routines

Function Definitions

```
void avfreewts(
                    long *p);

void avfreewts_(
                    long *p);
```

Parameters

Argument	I/O	Description
p	input	Pointer to a structure containing FFT setup.

Return Value

None: void function.

Description

A lookup table of complex numbers is computed for performing FFTs consisting of up to m complex samples. Optionally a second lookup table of complex numbers is computed for performing FFTs consisting of n real samples. A structure is created containing the sizes and base addresses of the generated lookup table(s). The address of that structure is returned as the value of p.

Restrictions and Notes

avrfftf

Real to Complex Forward FFT

Function Definitions

```
void avrfftf(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    float *rs0,
                    long   n,
                    long  *p);

void avrfftf_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    float *rs0,
                    long  *n,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	float or double vector.
rv1inc	input	long integer scalar (cv1 vector stride).
cvo	output	complex float or double vector.
cvoinc	input	long integer scalar (cvo vector stride).
rs0	output	float or double vector (Nyquist value).
n	input	long integer scalar (sample length).
p	input	Pointer to a structure containing FFT setup.

Return Value

None: void function.

Description

The forward real-to-complex FFT of the real input vector is computed and stored in the complex output vector.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.

4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avrffti

Complex to Real Inverse FFT

Function Definitions

```
void avrffti(
                    float *cv1,
                    long   cv1inc,
                    float *rs0,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    long  *p);

void avrffti_(
                    float *cv1,
                    long  *cv1inc,
                    float *rs0,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	complex float or double vector.
cv1inc	input	long integer scalar (cv1 vector stride).
rvo	output	float or double vector.
rvoinc	input	long integer scalar (rvo vector stride).
n	input	long integer scalar (sample length).
p	input	Pointer to a structure containing FFT setup.

Return Value

None: void function.

Description

The inverse complex-to-real FFT of the complex input vector is computed and stored in the real output vector.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.

4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avrft2fc

2D Real to Complex Forward FFT - Column Compact

Function Definitions

```
void avrft2fc(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p);

void avrft2fc_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1
p	input	pointer to FFT setup previously generated by calling avfftwts

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input matrix is computed and stored in the output matrix. The columns are assumed to be compact. The two dimensional forward FFT is performed as a series of forward real-to-complex FFTs across the rows, followed by forward complex-to-complex FFTs down the columns. The real-to-complex FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.

2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avrft2fr

2D Real to Complex Forward FFT - Row Compact

Function Definitions

```
void avrft2fr(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p);

void avrft2fr_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1
p	input	pointer to FFT setup previously generated by calling avfftwts

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional forward FFT is performed as a series of forward real-to-complex FFTs across the rows, followed by forward complex-to-complex FFTs down the columns. The real-to-complex FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.

2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avrft2ic

2D Complex to Real Inverse FFT - Column Compact

Function Definitions

```
void avrft2ic(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc,
                    long  *p);

void avrft2ic_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo
p	input	pointer to FFT setup previously generated by calling avfftwts

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input matrix is computed and stored in the output matrix. The columns are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs on input vector cvn and down the columns of cv1, followed by inverse complex-to-real FFTs across the rows. The complex-to-real FFTs each require a single complex Nyquist frequency input, which is taken from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.

2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avrft2ir

2D Complex to Real Inverse FFT - Row Compact

Function Definitions

```
void avrft2ir(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc,
                    long  *p);

void avrft2ir_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc,
                    long  *p);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo
p	input	pointer to FFT setup previously generated by calling avfftwts

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs on input vector cvn and down the columns of cv1, followed by inverse complex-to-real FFTs across the rows. The complex-to-real FFTs each require a single complex Nyquist frequency input, which is taken from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.

2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtcft2fc

Multi-threaded 2D Complex to Complex Fwd FFT - Column Compact

Function Definitions

```
void avtcft2fc(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtcft2fc(
                    float   *cv1,
                    float   *cvo,
                    long    *nr,
                    long    *nc,
                    long  * *p,
                    long    *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1
p	input	Pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional forward FFT is performed as a series of forward complex-to-complex FFTs across the rows, followed by complex-to-complex forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.

5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtcft2fr

Multi-threaded 2D Complex to complex Fwd FFT - Row Compact

Function Definitions

```
void avtcft2fr(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p,
                    long   nt);

void avtcft2fr_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p,
                    long  *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1
p	input	Pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The rows are assumed to be compact. The two dimensional forward FFT is performed as a series of forward complex-to-complex FFTs across the rows, followed by complex-to-complex forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.

5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtcft2ic

Multi-threaded 2D Complex to Complex Inv FFT - Column Compact

Function Definitions

```
void avtcft2ic(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long   p,
                    long   nt);

void avtcft2ic_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc,
                    long  *p,
                    long  *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1
p	input	Pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs down the columns, followed by complex-to-complex inverse FFTs across the rows.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.

5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtcft2ir

Multi-threaded 2D Complex to Complex Inv FFT - Row Compact

Function Definitions

```
void avtcft2ir(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtcft2ir_(
                    float  *cv1,
                    float  *cvo,
                    long   *nr,
                    long   *nc,
                    long  * *p,
                    long   *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1
p	input	Pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional inverse complex FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs down the columns, followed by complex-to-complex inverse FFTs across the rows.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.

5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtrft2fc

Multi-threaded 2D Real to Complex Fwd FFT - Column Compact

Function Definitions

```
void avtrft2fc(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtrft2fc_(
                    float  *rv1,
                    float  *cvn,
                    long   *cvninc,
                    float  *cvo,
                    long   *nr,
                    long   *nc,
                    long  **p,
                    long   *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1
p	input	pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input matrix is computed and stored in the output matrix. The columns are assumed to be compact. The two dimensional forward FFT is performed as a series of forward real-to-complex FFTs across the rows, followed by forward complex-to-complex FFTs down the columns. The real-to-complex FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtrft2fr

Multi-threaded 2D Real to Complex Fwd FFT - Row Compact

Function Definitions

```
void avtrft2fr(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtrft2fr_(
                    float  *rv1,
                    float  *cvn,
                    long   *cvninc,
                    float  *cvo,
                    long   *nr,
                    long   *nc,
                    long  * *p,
                    long   *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1
p	input	pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional forward FFT is performed as a series of forward real-to-complex FFTs across the rows, followed by forward complex-to-complex FFTs down the columns. The real-to-complex FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtrft2ic

Multi-threaded 2D Complex to Real Inv FFT - Column Compact

Function Definitions

```
void avtrft2ic(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtrft2ic_(
                    float  *cv1,
                    float  *cvn,
                    long   *cvninc,
                    float  *rvo,
                    long   *nr,
                    long   *nc,
                    long  * *p,
                    long   *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo
p	input	pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional real FFT is defined as a series of inverse complex-to-complex FFTs across the rows and input vector cvn, followed by inverse complex-to-real FFTs down the columns. The real FFTs each require a single real Nyquist frequency input, which is read from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

avtrft2ir

Multi-threaded 2D Complex to Real Inv FFT - Row Compact

Function Definitions

```
void avtrft2ir(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc,
                    long  *p,
                    long   nt);

void avtrft2ir_(
                    float  *cv1,
                    float  *cvn,
                    long   *cvninc,
                    float  *rvo,
                    long   *nr,
                    long   *nc,
                    long  **p,
                    long   *nt);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo
p	input	pointer to FFT setup previously generated by calling avfftwts
nt	input	Number of threads to use

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input matrix is computed and stored in the output matrix. The rows are assumed to be compact. The two dimensional inverse FFT is performed as a series of inverse complex-to-complex FFTs on input vector cvn and down the columns of cv1, followed by inverse complex-to-real FFTs across the rows. The complex-to-real FFTs each require a single complex Nyquist frequency input, which is taken from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

blkmanP

Apply a Blackman Window to a Real Vector

Function Definitions

```
void blkman(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void blkman_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void blkmand(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void blkmand_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies each element of the input vector rv1 by the Blackman window.

Restrictions and Notes

1. The strided elements in the input and output buffers must be of the same size.
2. The operation is performed faster if **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

blkman-harris

Blackman-Harris Window

Function Definitions

```
void blkman_harris(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    long   which);

void blkman_harris_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    long  *which);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process
which	input	selects the window type

Return Value

None: void function.

Description

This function generates a Blackman-Harris window, multiplying the window by the real input vector rv1 to produce the output vector rv2. The which parameter selects coefficients for the window from the following table:

which	a0	a1	a2	a4
0	0.44959	-0.49364	0.05677	0
1	0.42323	-0.49755	0.07922	0
2	0.40217	-0.49703	0.09892	-0.00188
3	0.35875	-0.48829	0.14128	-0.01168

Restrictions and Notes

1. The strided elements in the input and output buffers must be of the same size.
2. The operation is performed faster if **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

ccdotpP

Complex Conjugate Dot Product

Function Definitions

```
void ccdotp(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cso,
                    long   n);

void ccdotp_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cso,
                    long  *n);

void ccdotpd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cso,
                    long    n);

void ccdotpd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cso,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector 1
rv1inc	input	Vector increment for cv1
cv2	input	Complex input vector 2
cv2inc	input	Vector increment for cv2
cso	output	Complex scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

Second complex vector is multiplied by the conjugate of the first and summed into scalar cso.

Restrictions and Notes

1. The strided elements in the two input buffers must be equal to or greater than **n**.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cconvdP

Complex Convolution with Decimation

Function Definitions

```
void cconvd(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n0,
                    long   n2,
                    long   ndf);

void cconvd_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n0,
                    long  *n2,
                    long  *ndf);

void cconvdd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n0,
                    long    n2,
                    long    ndf);

void cconvdd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n0,
                    long   *n2,
                    long   *ndf);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cvo`	output	Complex output vector
`cvoinc`	input	Output vector increment
`no`	input	Number of points in output vector
`n2`	input	Number of points in vector cv2 (kernel)
`ndf`	input	Decimation factor

Return Value

None: void function.

Description

cconvd convolves with decimation the complex vector cv1 with the complex vector cv2.

Restrictions and Notes

1. The operation is performed faster when **no** and **n2** are a power of two.
2. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
3. This operation is supported in both single and double precision.
4. Complex buffers are in Complex Interleaved format.

ccorfP

Frequency Domain Cross Correlation

Function Definitions

```
void ccorf(
                    float *rv1,
                    float *rv2,
                    float *rvo,
                    long   n0,
                    long   n1);

void ccorf_(
                    float *rv1,
                    float *rv2,
                    float *rvo,
                    long  *n0,
                    long  *n1);

void ccorfd(
                    double *rv1,
                    double *rv2,
                    double *rvo,
                    long    n0,
                    long    n1);

void ccorfd_(
                    double *rv1,
                    double *rv2,
                    double *rvo,
                    long   *n0,
                    long   *n1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv2	input	Second real input vector
rvo	output	Real output vector
no	input	Number of points in output vector
n1	input	Number of points of vectors rv1 and rv2

Return Value

None: void function.

Description

Cross correlates, in the frequency domain, the inputs vectors rv1 and rv2 of size n1, and produces the output vector, rvo, with a size of no.

Restrictions and Notes

1. The strided elements in the two input buffers must be equal to or greater than **n1** in length.
2. The strided elements in the output buffer must be equal to or greater than **no** in length.
3. The operation is performed faster when **n1** and **no** are a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.
6. Complex buffers are in Complex Interleaved format.

ccortP

Time Domain Cross Correlation

Function Definitions

```
void ccort(
                    float *rv1,
                    float *rv2,
                    float *rvo,
                    long   n0,
                    long   n1);

void ccort_(
                    float *rv1,
                    float *rv2,
                    float *rvo,
                    long  *n0,
                    long  *n1);

void ccortd(
                    double *rv1,
                    double *rv2,
                    double *rvo,
                    long    n0,
                    long    n1);

void ccortd_(
                    double *rv1,
                    double *rv2,
                    double *rvo,
                    long   *n0,
                    long   *n1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv2	input	Second real input vector
rvo	output	Real output vector
no	input	Number of points in output vector
n1	input	Number of points in vectors rv1 and rv2

Return Value

None: void function.

Description

Cross correlates, in the time domain, the input vectors rv1 and rv2, with a size of n1, and produces the output vector, rvo, with a size of no.

Restrictions and Notes

1. The strided elements in the two input buffers must be equal to or greater than **n1** in length.
2. The strided elements in the output buffer must be equal to or greater than **no** in length.
3. The operation is performed faster when **n1** and **no** are a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.
6. Complex buffers are in Complex Interleaved format.

cdotprP

Complex Dot Product

Function Definitions

```
void cdotpr(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inv,
                    float *cso,
                    long   n);

void cdotpr_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inv,
                    float *cso,
                    long  *n);

void cdotprd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inv,
                    double *cso,
                    long    n);

void cdotprd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inv,
                    double *cso,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector 1
cv1inc	input	Vector increment for cv1
cv2	input	Complex input vector 2
cv2inc	input	Vector increment for cv2
cso	output	Complex scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

Two complex vectors are multiplied together and summed into the scalar cso.

Restrictions and Notes

1. The strided elements in the two input buffers must be equal to or greater than **n**.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cfftP

Complex to Complex FFT, in Place - Fwd and Inv

Function Definitions

```
void cfft(
                    float *cvio,
                    long   n,
                    long   flg);

void cfft_(
                    float *cvio,
                    long  *n,
                    long  *flg);

void cfftd(
                    double *cvio,
                    long    n,
                    long    flg);

void cfftd_(
                    double *cvio,
                    long   *n,
                    long   *flg);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cvio	in/output	Complex input vector
n	input	FFT size (number of complex points)
flg	input	Flag selecting transform direction flg = +1 => forward flg = -1 => inverse

Return Value

None: void function.

Description

The forward or inverse complex FFT of the input vector is computed in place.

Restrictions and Notes

1. The FFT is performed faster if the FFT length is a power of 2.
2. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
3. Complex buffers are in Complex Interleaved format.
4. This operation is supported in both single and double precision.

cfftfP

Complex to Complex Forward FFT

Function Definitions

```
void cfftf(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cfftf_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cfftfd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cfftfd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	FFT size (number of complex points)

Return Value

None: void function.

Description

The forward complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cfftfnP

Complex to Complex Fwd FFT (Non-Destructive)

Function Definitions

```
void cfftfn(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cfftfn_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cfftfnd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cfftfnd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	FFT size (number of complex points)

Return Value

None: void function.

Description

The forward complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cfftiP

Complex to Complex Inverse FFT

Function Definitions

```
void cffti(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cffti_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cfftid(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cfftid_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	IFFT size (number of complex points)

Return Value

None: void function.

Description

The inverse complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cfftinP

Complex to Complex Inv FFT (Non-Destructive)

Function Definitions

```
void cfftin(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cfftin_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cfftind(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cfftind_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	IFFT size (number of complex points)

Return Value

None: void function.

Description

The inverse complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cfftscP

Complex to Complex FFT Scale

Function Definitions

```
void cfftsc(
                    float *cvio,
                    long   n);

void cfftsc_(
                    float *cvio,
                    long  *n);

void cfftscd(
                    double *cvio,
                    long    n);

void cfftscd_(
                    double *cvio,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cvio	in/output	Complex input/output vector
n	input	Number of complex points in cvio

Return Value

None: void function.

Description

Each point in the input vector is divided by the real scalar n and stored back into the vector. This properly scales the output of a complex FFT.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cft2fcP

2D Complex to Complex Forward FFT - Column Compact

Function Definitions

```
void cft2fc(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc);

void cft2fc_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void cft2fcd(
                    double *cv1,
                    double *cvo,
                    long    nr,
                    long    nc);

void cft2fcd_(
                    double *cv1,
                    double *cvo,
                    long   *nr,
                    long   *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional forward FFT is defined as a series of forward FFTs across the rows, followed by forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cft2frP

2D Complex to Complex Forward FFT - Row Compact

Function Definitions

```
void cft2fr(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc);

void cft2fr_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void cft2frd(
                    double *cv1,
                    double *cvo,
                    long    nr,
                    long    nc);

void cft2fcd_(
                    double *cv1,
                    double *cvo,
                    long   *nr,
                    long   *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The rows are assumed to be compact. The two dimensional forward FFT is defined as a series of forward FFTs across the rows, followed by forward FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cft2icP

2D Complex to Complex Inverse FFT - Column Compact

Function Definitions

```
void cft2ic(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc);

void cft2ic_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void cft2icd(
                    double *cv1,
                    double *cvo,
                    long    nr,
                    long    nc);

void cft2icd_(
                    double *cv1,
                    double *cvo,
                    long   *nr,
                    long   *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional inverse FFT is defined as a series of inverse FFTs across the rows, followed by inverse FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

cft2irP

2D Complex to Complex Inverse FFT - Row Compact

Function Definitions

```
void cft2ir(
                    float *cv1,
                    float *cvo,
                    long   nr,
                    long   nc);

void cft2ir_(
                    float *cv1,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void cft2ird(
                    double *cv1,
                    double *cvo,
                    long    nr,
                    long    nc);

void cft2ird_(
                    double *cv1,
                    double *cvo,
                    long   *nr,
                    long   *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvo	output	Complex output vector
nr	input	Number of rows in cv1
nc	input	Number of columns in cv1

Return Value

None: void function.

Description

The two dimensional forward complex FFT of the input vector is computed and stored in the output vector. The rows are assumed to be compact. The two dimensional inverse FFT is defined as a series of inverse FFTs across the rows, followed by inverse FFTs down the columns.

Restrictions and Notes

1. The strided elements in the input and output buffer must be of the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT length is a power of 2.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

convdP

Convolution with Decimation

Function Definitions

```
void convd(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n0,
                    long   n2,
                    long   ndf);

void convd_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n0,
                    long  *n2,
                    long  *ndf);

void convdd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n0,
                    long    n2,
                    long    ndf);

void convdd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n0,
                    long   *n2,
                    long   *ndf);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Output vector increment
no	input	Number of points in output vector
n2	input	Number of points in vector rv2 (kernel)
ndf	input	Decimation factor

Return Value

None: void function.

Description

convd convolves with decimation the vector rv1 with the vector rv2.

Restrictions and Notes

1. The strided elements in the two input vectors must be equal to or greater than **n2** in length.
2. The strided elements in the output vector must be equal to or greater than **no** in length.
3. The operation is performed faster when **n2** and **no** are a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.
6. Complex buffers are in Complex Interleaved format.

cpows

Complex Vector Power with Add

Function Definitions

```
void cpows(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    float *rv2,
                    long   rv2inc,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cpows_(
                    float *cv1,
                    long  *cv1inc,
                    float *rs1,
                    float *rv2,
                    long  *rv2inc,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
rs1	input	Real scalar 1
rv2	input	Real input vector
rv2inc	input	Vector increment for rv2
rs2	input	Real scalar 2
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The magnitude squared of complex vector cv1 is scaled by real rs1 and summed with real vector rv1 scaled by rs2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

crvdivP

Complex Vector Real Vector Divide

Function Definitions

```
void crvdiv(
                    float *cv1,
                    long   cv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void crvdiv_(
                    float *cv1,
                    long  *cv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long   n);

void crvdivd(
                    double *cv1,
                    long    cv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void crvdivd_(
                    double *cv1,
                    long   *cv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long    n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector
`cv1inc`	input	Input vector increment
`rv2`	input	Real input vector
`rv2inc`	input	Real vector increment
`cvo`	output	Complex output vector
`cvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector, cv1, is divided by the corresponding element of the real vector, rv2, and copied to the complex output vector, cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

crvmulP

Complex Vector Real Vector Multiply

Function Definitions

```
void crvmul(
                    float *cv1,
                    long   cv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void crvmul_(
                    float *cv1,
                    long  *cv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long   n);

void crvmuld(
                    double *cv1,
                    long    cv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void crvmuld_(
                    double *cv1,
                    long   *cv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long    n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector
`cv1inc`	input	Complex vector increment
`rv2`	input	Real input vector
`rv2inc`	input	Real vector increment
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector cvl is multiplied by the corresponding element of the real vector, rv2, and copied to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cspecP

Accumulating Cross Spectrum

Function Definitions

```
void cspec(
                    float *cv1,
                    float *cv2,
                    float *cvo,
                    long   n);

void cspec_(
                    float *cv1,
                    float *cv2,
                    float *cvo,
                    long  *n);

void cspecd(
                    double *cv1,
                    double *cv2,
                    double *cvo,
                    long    n);

void cspecd_(
                    double *cv1,
                    double *cv2,
                    double *cvo,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector 1
cv2	input	Complex input vector 2
cvo	in/output	Complex input/output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The complex conjugate of the first input vector, cv1, is multiplied by the second input vector, cv2, and summed with the i/o vector cvo. (This routine is included only for backwards compatibility with other subroutine libraries. For new programs, use cvcmla.)

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

csve

Complex Sum Vector Elements

Function Definitions

```
void csve(
                    float *cv1,
                    long   cv1inc,
                    float *cs1,
                    long   n);

void csve_(
                    float *cv1,
                    long  *cv1inc,
                    float *cs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cs1	output	Location to deposit complex scalar result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function sums all complex elements in the vector cv1, storing the complex result at *cs1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvabsP

Complex Vector Absolute Value

Function Definitions

```
void cvabs(
                    float *cv1,
                    long   cv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvabs_(
                    float *cv1,
                    long  *cv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvabsd(
                    double *cv1,
                    long    cv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void cvabsd_(
                    double *cv1,
                    long   *cv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Complex vector increment
rvo	output	Real output vector
rvoinc	input	Real vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

The absolute value of each element of the complex input vector is copied to the real output vector, rvo. The absolute value is computed from the square root of the magnitude squared of each element.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvabserr

Complex Absolute Error

Function Definitions

```
void cvabserr(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long   n);

void cvabserr_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cv2	input	Input complex vector
cv2inc	input	Input vector increment
avg	output	Average error
max	output	Maximum error
i	output	Index where max occurred
n	input	Number of samples to process

Return Value

None: void function.

Description

The absolute error computes |rv1k - rv2k| for all k elements in each vector. The average of this value is computed and stored at *avg. The maximum error value is stored at *max, and the index to the maximum value is stored at *i.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvaddP

Complex Vector Add

Function Definitions

```
void cvadd(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    long   n);

void cvadd_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    long  *n);

void cvaddd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    long    n);

void cvaddd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector 1
cv1inc	input	Vector increment for cv1
cv2	input	Complex input vector 2
cv2inc	input	Vector increment for cv2
cvo	output	Complex output vector
cvoinc	input	Vector increment for cvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The elements of the complex input vectors are added together and stored to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvaddsub

Complex Add/Subtract

Function Definitions

```
void cvaddsub(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvs,
                    long   cvsinc,
                    float *cvd,
                    long   cvdinc,
                    long   n);

void cvaddsub_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvs,
                    long  *cvsinc,
                    float *cvd,
                    long  *cvdinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cv2	input	Input complex vector
cv2inc	input	Input vector increment
cvs	output	complex sum
incs	input	stride for cvs
cvd	output	complex difference
incd	input	stride for cvd
n	input	Number of samples to process

Return Value

None: void function.

Description

This function adds elements from cv1 and cv2 placing the result in cvs. Concurrently it computes cv1 - cv2, placing the difference in cvd. This function may be used in-place.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvavg

Complex Average

Function Definitions

```
void cvavg(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvavg_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cv2	input	Input complex vector
cv2inc	input	Input vector increment
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the average of corresponding complex elements of cv1 and cv2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvavgnegy

Complex Average x, -y

Function Definitions

```
void cvavgnegy(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvavgnegy_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cv2	input	Input complex vector
cv2inc	input	Input vector increment
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the average of corresponding complex elements of cv1 and -cv2

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvaxyplusz

Complex aXY + Z

Function Definitions

```
void cvaxyplusz(
                    float       *cvx,
                    long         cvxinc,
                    float       *cvy,
                    long         cvyinc,
                    float       *cvz,
                    long         cvzinc,
                    float       *a,
                    float       *vout,
long                     voutinc,
                    long         n);

void cvaxyplusz_(
                    float       *cvx,
                    long        *cvxinc,
                    float       *cvy,
                    long        *cvyinc,
                    float       *cvz,
                    long        *cvzinc,
                    float       *a,
                    float       *vout,
long                 *   voutinc,
                    long        *n);
```

Parameters

Argument	I/O	Description
cvx	input	Input complex vector
incx	input	Input vector increment
cvy	input	Input complex vector
incy	input	Input vector increment
cvz	input	Input complex vector
incz	input	Input vector increment
a	input	Input complex scalar
vout	output	Output complex vector
inco	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes aXY + Z for a complex scalar a and complex vectors X, Y and Z.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvceil

Complex Ceiling

Function Definitions

```
void cvceil(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvceil_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input complex vector
rv1inc	input	Input vector increment
rvo	output	Output complex vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the ceil for each input complex vector element. The ceil of x is x if x is an integer, otherwise it is the truncated value of x+1. For example, ceil(2) = 2; ceil (-3.1) = -3, ceil (4.1) = 5.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvclip

Complex Clip

Function Definitions

```
void cvclip(
                    float *cv1,
                    long   cv1inc,
                    float *lo,
                    float *hi,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvclip_(
                    float *cv1,
                    long  *cv1inc,
                    float *lo,
                    float *hi,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
lo	input	Pointer to low limit
hi	input	Pointer to high limit
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function clips the input vector: real or imaginary values < *lo are replaced with *lo, and real or imaginary values > *hi are replaced with *hi. Note that *lo and *hi are real scalar that are compared against both the real and imaginary parts of the input vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvcliph

Complex Clip High

Function Definitions

```
void cvcliph(
                    float *cv1,
                    long   cv1inc,
                    float *hi,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcliph_(
                    float *cv1,
                    long  *cv1inc,
                    float *hi,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
hi	input	Pointer to high limit
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function clips the input vector: real or imaginary values < *hi are replaced with *hi. Note that *hi is a real scalar that is compared against both the real and imaginary portions of the input vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvclipl

Complex Clip Low

Function Definitions

```
void cvclipl(
                    float *cv1,
                    long   cv1inc,
                    float *lo,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvclipl_(
                    float *cv1,
                    long  *cv1inc,
                    float *lo,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
lo	input	Pointer to low limit
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function clips the input vector: real or imaginary values < *lo are replaced with *lo. Note that *lo is a real scalar that is compared against both the real and imaginary parts of the input vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvcmlP

Complex Vector Conjugate Multiply

Function Definitions

```
void cvcml(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcml_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvcmld(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvcmld_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Elements of the second complex vector are multiplied by the complex conjugate of the first complex vector. The results are written to the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvcmlaP

Complex Vector Conjugate Multiply with Add

Function Definitions

```
void cvcmla(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cv3,
                    long   cv3inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcmla_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cv3,
                    long  *cv3inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvcmlad(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cv3,
                    long   cv3inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcmlad_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cv3,
                    long  *cv3inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cv3`	input	Complex input vector 3
`cv3inc`	input	Vector increment for cv3
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

The complex conjugate of vector cv1 is multiplied by the complex vector cv2, that result is added to the elements from complex vector cv3 and stored in complex vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvcmpXX

Compare Complex Vectors

Function Definitions

```
void cvcmpeq(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmpeq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvcmpne(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmpne_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvcmpge(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmpge_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

```
void cvcmpgt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmpgt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvcmple(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmple_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvcmplt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcmplt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Input complex vector
`rv1inc`	input	Input vector increment
`rv2`	input	Input complex vector
`rv2inc`	input	Input vector increment
`rvo`	output	Output vector
`rvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

This function compares the magnitude squared of corresponding elements of rv1 and rv2, leaving a 32 bit long flag value in the corresponding elements of rv3. If the condition is true, the resulting flag value is 0xffff_ffff. If the resulting condition is false, the resulting flag value is 0x0000_0000. The resulting flag vectors may then be operated upon by other long integer vector functions if desired.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvcombP

Form Complex Vector from Two Real Vectors

Function Definitions

```
void cvcomb(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvcomb_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvcombd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void cvcombd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector (r)
`rv1inc`	input	Vector increment for rv1
`rv2`	input	Real input vector (i)
`rv2inc`	input	Vector increment for rv2
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Sets the desired elements of the complex array to a complex value with the real part equal to elements of the first vector and the imaginary parts equal to elements of the second vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvconjP

Complex Vector Conjugate

Function Definitions

```
void cvconj(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvconj_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvconjd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvconjd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
cvo	output	Complex output vector
cvoinc	input	Vector increment for cbo
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector is copied to the complex output vector, cvo, with the imaginary part negated (complex conjugate).

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvcsmlP

Complex Vector Complex Scalar Multiply

Function Definitions

```
void cvcsml(
                    float *cv1,
                    long   cv1inc,
                    float *cs1,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcsml_(
                    float *cv1,
                    long  *cv1inc,
                    float *cs1,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvcsmld(
                    double *cv1,
                    long    cv1inc,
                    double *cs1,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvcsmld_(
                    double *cv1,
                    long   *cv1inc,
                    double *cs1,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
cs1	input	Complex scalar
cvo	output	Complex output vector
cvoinc	input	Vector increment for cvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector is multiplied by the complex scalar cs1 and is then copied to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvcvrsmlP

Complex Vector Complex Vector Real Scalar Multiply

Function Definitions

```
void cvcvrsml(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *rs1,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvcvrsml_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *rs1,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvcvrsmld(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *rs1,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvcvrsmld_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *rs1,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv1inc`	input	Vector increment for cv2
`rs1`	input	Real scalar
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Elements of the two complex input vectors are multiplied times each other then times the real scalar rs1. The result is copied to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvdivP

Complex Vector Complex Vector Division

Function Definitions

```
void cvdiv(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvdiv_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvdivd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvdivd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Complex vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Complex vector increment for cv2
`cvo`	output	Complex output vector
`cvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Elements of the first complex input vector, cv1, are divided by elements of the second complex input vector, cv2, and copied to the complex output vector, cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvexpP

Complex Vector Exponential

Function Definitions

```
void cvexp(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvexp_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvexpd(
                    double *rv1,
                    long    rv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvexpd_(
                    double *rv1,
                    long   *rv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cv1inc	input	Increment for input vector
cvo	output	Complex output vector
cvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes complex exponential of each element of real input vector, with input values expressed in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvfillP

Set a Complex Vector to a Complex Scalar

Function Definitions

```
void cvfill(
                    float *cs1,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvfill_(
                    float *cs1,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvfilld(
                    double *cs1,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvfilld_(
                    double *cs1,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cs1	input	Complex scalar
cvo	output	Complex output vector
cvoinc	input	Increment for output vector
n	input	Number of points in output vector

Return Value

None: void function.

Description

Fills vector cvo with the scalar cs1.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.

3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvfindlastXX

Find Element in a Complex Vector

Function Definitions

```
void cvfindlasteq(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindlasteq_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindlastne(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindlastne_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindlastgt(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindlastgt_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindlastge(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindlastge_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);
```

```
void cvfindlastle(
                        float *cv1,
                        long   cv1inc,
                        float *k,
                        long  *i,
                        long   n);

void cvfindlastle_(
                        float *cv1,
                        long  *cv1inc,
                        float *k,
                        long  *i,
                        long  *n);

void cvfindlastlt(
                        float *cv1,
                        long   cv1inc,
                        float *k,
                        long  *i,
                        long   n);

void cvfindlastlt_(
                        float *cv1,
                        long  *cv1inc,
                        float *k,
                        long  *i,
                        long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
k	input	complex value for which to search
i	output	Resulting index value
n	input	Number of samples to process

Return Value

None: void function.

Description

These functions are similar to cvfindXX, except that the search proceeds from right to left. See cvfindXX.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.

3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvfindXX

Find Element in a Complex Vector

Function Definitions

```
void cvfindeq(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindeq_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindne(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindne_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindge(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindge_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void cvfindgt(
                    float *cv1,
                    long   cv1inc,
                    float *k,
                    long  *i,
                    long   n);

void cvfindgt_(
                    float *cv1,
                    long  *cv1inc,
                    float *k,
                    long  *i,
                    long  *n);
```

```
void cvfindle(
                        float *cv1,
                        long   cv1inc,
                        float *k,
                        long  *i,
                        long   n);

void cvfindle_(
                        float *cv1,
                        long  *cv1inc,
                        float *k,
                        long  *i,
                        long  *n);

void cvfindlt(
                        float *cv1,
                        long   cv1inc,
                        float *k,
                        long  *i,
                        long   n);

void cvfindlt_(
                        float *cv1,
                        long  *cv1inc,
                        float *k,
                        long  *i,
                        long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
k	input	complex scalar value for which to search
i	output	Resulting index value
n	input	Number of samples to process

Return Value

None: void function.

Description

cvfindeq: This function searches the real vector cv1 from left to right until a value whose magnitude squared is equal to the magnitude squared of k is located. If no value matches, this function sets *i to -1. Otherwise this sets *i to the index of the value meeting the search criteria.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvfloor

Complex Floor

Function Definitions

```
void cvfloor(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvfloor_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input complex vector
cv1inc	input	Input vector increment
rvo	output	Output complex vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the floor for each input vector element. The floor of x is x if x is an integer, otherwise it is the truncated value of x-1. For example, floor(2) = 2; floor (-3.1) = -4, floor (4.1) = 4.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvintegrate

Discrete Complex Integrator

Function Definitions

```
void cvintegrate(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvintegrate_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

For each element cvok in the output vector, this function computes the sum of the first k entries in the input vector cv1. This function is sometimes called a "running sum".

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvmaP

Complex or Conjugate Vector Multiply with Add

Function Definitions

```
void cvma(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cv3,
                    long   cv3inc,
                    float *cvo,
                    long   cvoinc,
                    long   n,
                    long   flg);

void cvma_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cv3,
                    long  *cv3inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n,
                    long  *flg);

void cvmad(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cv3,
                    long    cv3inc,
                    double *cvo,
                    long    cvoinc,
                    long    n,
                    long    flg);

void cvmad_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cv3,
                    long   *cv3inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n,
                    long   *flg);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cv3`	input	Complex input vector 3
`cv3inc`	input	Vector increment for cv3
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process
`flg`	input	Flag to specify complex or conjugate multiply flg >= 0 => Complex Multiply flg < 0 => Conjugate Multiply

Return Value

None: void function.

Description

Elements of the first two input complex vectors are multiplied together as specified by flg and added to the third complex input vector. Complex results are written to the complex output vector

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmagsP

Complex Vector Magnitude Squared

Function Definitions

```
void cvmags(
                    float *cv1,
                    long   cv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvmags_(
                    float *cv1,
                    long  *cv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvmagsd(
                    double *cv1,
                    long    cv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void cvmagsd_(
                    double *cv1,
                    long   *cv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The squares of the real and imaginary parts of each element in the complex input vector are summed and returned in the corresponding element of the real output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmerge

Merge Real, Imag into Complex

Function Definitions

```
void cvmerge(
                    float *rvr,
                    long   rvrinc,
                    float *rvi,
                    long   rviinc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvmerge_(
                    float *rvr,
                    long  *rvrinc,
                    float *rvi,
                    long  *rviinc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rvr	input	Input real vector
incr	input	Input vector increment
rvi	input	Input real vector
inci	input	Input vector increment
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function forms a complex output vector by taking elements from rvr and rvi. Elements from rvr become the real elements of the output vector, elements from rvi become the imaginary elements of the output vector. This function is the inverse of cvsplit.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvmexpP

Complex Vector Exponential with Multiply

Function Definitions

```
void cvmexp(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvmexp_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvmexpd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvmexpd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Real vector increment 1
`rv2`	input	Second real input vector
`rv2inc`	input	Second input vector increment
`cvo`	output	Complex vector output
`cvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

The cosine of each element of the real input vector, rv1, is scaled by the corresponding element of rv2 and assigned to the real part of the complex output vector, cvo. The sine of each element of the real input vector, rv1, is scaled by rv2 and assigned to the imaginary part of the complex output vector, cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmgsaP

Complex Vector Magnitude Squared with Add

Function Definitions

```
void cvmgsa(
                    float *cv1,
                    long   cv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvmgsa_(
                    float *cv1,
                    long  *cv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvmgsad(
                    double *cv1,
                    long    cv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void cvmgsad_(
                    double *cv1,
                    long   *cv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector
`cv1inc`	input	Complex vector increment
`rv2`	input	Real input vector
`rv2inc`	input	Real vector increment
`rvo`	output	Real output vector
`rvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector cv1 is multiplied by its own conjugate and the corresponding element of the real vector rv2 is added to it. The result is placed in the real vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmlP

Complex Vector Multiply

Function Definitions

```
void cvml(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvml_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvmld(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvmld_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Elements of the two input complex vectors are multiplied together. Complex results are written to the complex output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmlaP

Complex Vector Multiply with Add

Function Definitions

```
void cvmla(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cv3,
                    long   cv3inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvmla_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cv3,
                    long  *cv3inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvmlad(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cv3,
                    long    cv3inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvmlad_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cv3,
                    long   *cv3inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment for cv1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment for cv2
`cv3`	input	Complex input vector 3
`cv3inc`	input	Vector increment for cv3
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Elements of the first two input complex vectors are multiplied together and added to the third complex input vector. Complex results are written to the complex output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmovP

Complex Vector Move

Function Definitions

```
void cvmov(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvmov_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvmovd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvmovd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
cvo	output	Complex output vector
cvoinc	input	Vector increment for cvo
n	input	Number of points to move

Return Value

None: void function.

Description

Elements from the input vector cv1 are copied to the output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvmulP

Complex Vector Multiply or Conjugate Multiply

Function Definitions

```
void cvmul(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n,
                    long   flg);

void cvmul_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n,
                    long  *flg);

void cvmuld(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n,
                    long    flg);

void cvmuld_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n,
                    long   *flg);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector 1
`cv1inc`	input	Vector increment 1
`cv2`	input	Complex input vector 2
`cv2inc`	input	Vector increment 2
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for cvo
`n`	input	Number of samples to process
`flg`	input	Flag to specify complex or conjugate multiply flg >= 0 => Complex Multiply flg < 0 => Conjugate Multiply

Return Value

None: void function.

Description

The two complex vectors are multiplied and the result written to the output vector. The flg parameter controls whether or not the complex conjugate of the first vector is used.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvnegP

Complex Vector Negate

Function Definitions

```
void cvneg(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvneg_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvnegd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvnegd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
cvo	output	Complex output vector
cvoinc	input	Vector increment for cvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector cv1 is negated and copied to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvolsave

Complex Overlap and Save

Function Definitions

```
void cvolsave(
                    float *cv1,
                    long   cv1inc,
                    float *save,
                    long   slength,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvolsave_(
                    float *cv1,
                    long  *cv1inc,
                    float *save,
                    long  *slength,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
save	input	Address of complex save buffer
slen	input	Length of save buffer
cvo	output	Complex output vector
cvoinc	input	Output vector stride
n	input	length of input vector cv1

Return Value

None: void function.

Description

This function performs a complex overlap and save function:

- The first slen samples in the output vector are copied from the save buffer
- The next n samples of the output vector are copied from the input vector
- The last slen samples of the input buffer are copied into the save buffer

The resulting output vector is slen + n elements long. The input vector length must be greater than or equal to slen. The save buffer must be capable of containing 2*slen floats.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvphasP

Complex Vector Phase

Function Definitions

```
void cvphas(
                    float *cv1,
                    long   cv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvphas_(
                    float *cv1,
                    long  *cv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void cvphasd(
                    double *cv1,
                    long    cv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void cvphasd_(
                    double *cv1,
                    long   *cv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

The phase or argument (theta from polar form) of each element in the complex input vector is copied to the real output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvrcipP

Complex Vector Reciprocal

Function Definitions

```
void cvrcip(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvrcip_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvrcipd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvrcipd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

The reciprocal of the elements of the complex input vector cv1 is copied to the complex output vector cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvrealP

Make a Complex Vector from a Real Vector

Function Definitions

```
void cvreal(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvreal_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvreald(
                    double *rv1,
                    long    rv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvreald_(
                    double *rv1,
                    long   *rv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

A complex vector, cvo, is created from a real vector rv1. The imaginary part of cvo is set to zero.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvrelerr

Complex Relative Error

Function Definitions

```
void cvrelerr(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long   n);

void cvrelerr_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cv2	input	Input complex vector
cv2inc	input	Input vector increment
avg	output	Average error
max	output	Maximum error
i	output	Index where max occurred
n	input	Number of samples to process

Return Value

None: void function.

Description

The function computes the complex relative error for all k elements in each vector. The average of this value is computed and stored at *avg. The maximum error value is stored at *max, and the index to the maximum value is stored at *i.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvround

Complex Round

Function Definitions

```
void cvround(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvround_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input complex vector
rv1inc	input	Input vector increment
rvo	output	Output complex vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function rounds each input vector element to the nearest integer value. For example, $round(2) = 2$; $round(-3.7) = -4$, $round(4.1) = 4$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvrvrs

Complex Reverse

Function Definitions

```
void cvrvrs(
                    float *rv1,
                    long   rv1inc,
                    long   n);

void cvrvrs_(
                    float *rv1,
                    long  *rv1inc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	in/output	Input/output complex vector
rv1inc	input	Input vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function reverses the order of all elements in the specified vector.

Restrictions and Notes

1. The strided elements in the input/output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input/output vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

cvsadd

Complex Scalar Add

Function Definitions

```
void cvsadd(
                    float *cv1,
                    long   cv1inc,
                    float *cs1,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvsadd_(
                    float *cv1,
                    long  *cv1inc,
                    float *cs1,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cs1	input	Pointer to complex scalar
cvo	output	Output complex vector
cvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function adds the complex scalar at *cs1 to each complex element in the vector cv1 producing the complex vector result at *cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvsmaP

Scale Complex Vector and Add to Second Complex Vector

Function Definitions

```
void cvsma(
                    float *cv1,
                    long   cv1inc,
                    float *cs1,
                    float *cv2,
                    long   cv2inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvsma_(
                    float *cv1,
                    long  *cv1inc,
                    float *cs1,
                    float *cv2,
                    long  *cv2inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvsmad(
                    double *cv1,
                    long    cv1inc,
                    double *cs1,
                    double *cv2,
                    long    cv2inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvsmad_(
                    double *cv1,
                    long   *cv1inc,
                    double *cs1,
                    double *cv2,
                    long   *cv2inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`cv1`	input	Complex input vector
`cv1inc`	input	Complex vector increment for cv1
`cs1`	input	Complex scalar
`cv2`	input	Complex input vector
`cv2inc`	input	Complex vector increment for cv2
`cvo`	output	Complex output vector
`cvoinc`	input	Output vector increment
`n`	input	Number of points to generate

Return Value

None: void function.

Description

Each element of the complex input vector, cv1, is multiplied by the complex scalar, cs1, and then added with each element from complex vector, cv2, then finally, copied to the complex output vector, cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvsmulP

Complex Vector Scalar Multiply

Function Definitions

```
void cvsmul(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvsmul_(
                    float *cv1,
                    long  *cv1inc,
                    float *rs1,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvsmuld(
                    double *cv1,
                    long    cv1inc,
                    double *rs1,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvsmuld_(
                    double *cv1,
                    long   *cv1inc,
                    double *rs1,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
rs1	input	Real scalar
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of points to generate

Return Value

None: void function.

Description

Each element of the complex input vector, cv1, is multiplied by the real scalar, rs1, and then copied to the complex output vector, cvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvsplit

Split Complex into Real, Imag

Function Definitions

```
void cvsplit(
                    float *cvz,
                    long   cvzinc,
                    float *cvr,
                    long   cvrinc,
                    float *cvi,
                    long   cviinc,
                    long   n);

void cvsplit_(
                    float *cvz,
                    long  *cvzinc,
                    float *cvr,
                    long  *cvrinc,
                    float *cvi,
                    long  *cviinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvz	input	Input complex vector
incz	input	Input vector increment
rvr	output	Output real vector
incr	input	Input vector increment
rvi	output	Output real vector
inci	input	Input vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function splits the complex input vector cvz into real two real vectors; vector rvr contains the real components, vector rvi contains the imaginary components. This function is the inverse of cvmerge.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision)..
5. Complex buffers are in Complex Interleaved format.

cvsqrtP

Complex Vector Square Root.

Function Definitions

```
void cvsqrt(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void cvsqrt_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void cvsqrtd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void cvsqrtd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of points to generate

Return Value

None: void function.

Description

The square root of each element of the complex input vector cv1 is copied to the complex output vector cvo. The vector is first converted from rectangular to polar form, i.e., magnitude and angle. The complex square root is computed by taking the square root of the magnitude and half the angle, then converting back to rectangular form.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvsubP

Complex Vector Subtract

Function Definitions

```
void cvsub(
                    float *cv1,
                    long   cv1inc,
                    float *cv2,
                    long   cv2inc,
                    long   n);

void cvsub_(
                    float *cv1,
                    long  *cv1inc,
                    float *cv2,
                    long  *cv2inc,
                    long  *n);

void cvsubd(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inc,
                    long    n);

void cvsubd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cv2	input	Second complex input vector
cv2inc	input	Second input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of points to generate

Return Value

None: void function.

Description

The elements of the complex input vector cv2 are subtracted from the elements in cv1 and stored to cv2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

cvtrunc

Complex Truncate

Function Definitions

```
void cvtrunc(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void cvtrunc_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input complex vector
rv1inc	input	Input vector increment
rvo	output	Output complex vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function truncates each to an integer value. The truncation function discards the fractional portion of each value. For example, trunc(2) = 2; trunc (-3.1) = -3, trunc (4.1) = 4.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

deq22P

Difference Equation, 2 Poles, 2 Zeroes

Function Definitions

```
void deq22(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void deq22_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void deq22d(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void deq22d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	First real input vector
rv1inc	input	First input vector increment
rv2	input	Second real input vector Five filter coefficients with increment of one.
rvo	output	Real output vector
rvoinc	output	Output vector increment
n	input	Number of points to output

Return Value

None: void function.

Description

Performs a 2-pole, 2-zero recursive filter difference on vector rv1 using the vector rv2 as the filter. The result is stored in vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

5. Complex buffers are in Complex Interleaved format.

dolph-chebychev

Dolph-Chebychev Window

Function Definitions

```
void dolph_chebychev(
                     float *rv1,
                     long   rv1inc,
                     float *rvo,
                     long   rvoinc,
                     long   n,
                     float  alpha);

void dolph_chebychev_(
                     float *rv1,
                     long  *rv1inc,
                     float *rvo,
                     long  *rvoinc,
                     long  *n,
                     float *alpha);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process
alpha	input	alpha parameter for window computation

Return Value

None: void function.

Description

This function generates a Dolph-Chebychev window, multiplying the window by the real input vector rv1 to produce the output vector rv2.

Restrictions and Notes

1. The strided elements in the input and output buffers must be of the same size.
2. The operation is performed faster if the **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

‘

dotprP

Real Dot Product

Function Definitions

```
void dotpr(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rso,
                    long   n);

void dotpr_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rso,
                    float *n);

void dotprd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rso,
                    long    n);

void dotprd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rso,
                    float  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rso	output	Real scalar output
n	input	Number of points to output

Return Value

None: void function.

Description

Two real vectors, rv1 and rv2, are multiplied together and summed into a scalar rso.

Restrictions and Notes

1. The strided elements in the two input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

envelP

Vector Envelope

Function Definitions

```
void envel(
                    float *rv1,
                    float *rvo,
                    long   n);

void envel_(
                    float *rv1,
                    float *rvo,
                    long  *n);

void enveld(
                    double *rv1,
                    double *rvo,
                    long    n);

void enveld_(
                    double *rv1,
                    double *rvo,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rvo	output	Real output vector
n	input	FFT size (number of real points)

Return Value

None: void function.

Description

The envelope of the input vector is computed and stored in the output vector. The envelope is computed from the square root of the sum of the square of the vector elements plus the square of the Hilbert transform of the vector.

Restrictions and Notes

1. Both input and output buffers are of type compact.
2. Both input and output buffers must be bigger than **n**.
3. The operation is performed faster if the **n** is a power of two.
4. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.

f3x3P

Filter 3x3 Convolution

Function Definitions

```
void f3x3(
                    float *rm1,
                    long   m,
                    long   n,
                    float *rm2,
                    float *rmo);

void f3x3_(
                    float *rm1,
                    long  *m,
                    long  *n,
                    float *rm2,
                    float *rmo);

void f3x3d(
                    double *rm1,
                    long    m,
                    long    n,
                    double *rm2,
                    double *rmo);

void f3x3d_(
                    double *rm1,
                    long   *m,
                    long   *n,
                    double *rm2,
                    double *rmo);
```

Parameters

Argument	I/O	Description
rm1	input	Input image
m	input	Number of rows in input and output
n	input	Number of columns in input and output
rm2	input	3x3 input matrix of coefficients
rmo	output	Output image

Return Value

None: void function.

Description

Filter an m-by-n image by performing a two-dimensional convolution with a 3-by-3 kernel. The perimeter of the output image is padded with a border of zeros of width one.

Restrictions and Notes

1. The operation is performed faster when **m** and **n** are of power of two.
2. The input and output matrices are of type compact.
3. The operation is performed faster when the input and output matrix are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. The number of rows must be greater than or equal to 3.
6. The number of columns must be greater than 3 and even.

f5x5P

Filter 5x5 Convolution

Function Definitions

```
void f5x5(
                    float *rm1,
                    long   m,
                    long   n,
                    float *rm2,
                    float *rmo);

void f5x5_(
                    float *rm1,
                    long  *m,
                    long  *n,
                    float *rm2,
                    float *rmo);

void f5x5d(
                    double *rm1,
                    long    m,
                    long    n,
                    double *rm2,
                    double *rmo);

void f5x5d_(
                    double *rm1,
                    long   *m,
                    long   *n,
                    double *rm2,
                    double *rmo);
```

Parameters

Argument	I/O	Description
rm1	input	Input image
m	input	Number of rows in input and output
n	input	Number of columns in input and output
rm2	input	5x5 input matrix of coefficients
rmo	output	Output image

Return Value

None: void function.

Description

Filter an m-by-n image by performing a two-dimensional convolution with a 5-by-5 kernel. The perimeter of the output image is padded with a border of zeros of width two.

Restrictions and Notes

1. The operation is performed faster when **m** and **n** are of power of two.
2. The input and output matrices are of type compact.
3. The operation is performed faster when the input and output matrix are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. The number of rows must be greater than or equal to 5.
6. The number of columns must be greater than 5 and even.

fftwtsP

Create FFT Complex Exponential Tables

Function Definitions

```
void fftwts(
                    float *cvo,
                    long   m,
                    long   n);

void fftwts_(
                    float *cvo,
                    long  *m,
                    long  *n);

void fftwtsd(
                    double *cvo,
                    long    m,
                    long    n);

void fftwtsd_(
                    double *cvo,
                    long   *m,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cvo	output	Complex output vector
m	input	Max FFT size (number of complex points)
n	input	FFT size (number of complex points)

Return Value

None: void function.

Description

A table of complex exponentials is computed in output vector cvo, for use by FFT and Hanning/Hamming windowing functions.

Restrictions and Notes

1. The FFTs are performed faster if the FFT lengths are a power of two.

None.

fix2nP

Vector Fix to Two-byte Integer and Round

Function Definitions

```
void fix2n(
                    float *rv1,
                    long   rv1inc,
                    short *sv0,
                    long   sv0inc,
                    long   n);

void fix2n_(
                    float *rv1,
                    long  *rv1inc,
                    short *sv0,
                    long  *sv0inc,
                    long  *n);

void fix2nd(
                    double *rv1,
                    long    rv1inc,
                    short  *sv0,
                    long    sv0inc,
                    long    n);

void fix2nd_(
                    double *rv1,
                    long   *rv1inc,
                    short  *sv0,
                    long   *sv0inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
svo	output	Two-byte integer output vector
svoinc	input	Output vector increment
n	input	Number of points to process

Return Value

None: void function.

Description

Converts the real input vector rv1 to integer by rounding and placing the result in a two-byte integer output vector svo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fix4P

Vector Fix to Four-byte Integer and Truncate

Function Definitions

```
void fix4(
                    float *rv1,
                    long   rv1inc,
                    long  *iv0,
                    long   iv0inc,
                    long   n);

void fix4_(
                    float *rv1,
                    long  *rv1inc,
                    long  *iv0,
                    long  *iv0inc,
                    long  *n);

void fix4d(
                    double *rv1,
                    long    rv1inc,
                    long   *iv0,
                    long    iv0inc,
                    long    n);

void fix4d_(
                    double *rv1,
                    long   *rv1inc,
                    long   *iv0,
                    long   *iv0inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Real vector increment
ivo	output	Four-byte integer output vector
ivoinc	input	Output vector increment
n	input	Number of points to process

Return Value

None: void function.

Description

Converts the real input vector rv1 to integer by truncating, and places the result in four-byte integer output vector ivo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fix4nP

Vector Fix to Four-byte Integer and Round

Function Definitions

```
void fix4n(
                    float *rv1,
                    long   rv1inc,
                    long  *iv0,
                    long   iv0inc,
                    long   n);

void fix4n_(
                    float *rv1,
                    long  *rv1inc,
                    long  *iv0,
                    long  *iv0inc,
                    long  *n);

void fix4nd(
                    double *rv1,
                    long    rv1inc,
                    long   *iv0,
                    long    iv0inc,
                    long    n);

void fix4nd_(
                    double *rv1,
                    long   *rv1inc,
                    long   *iv0,
                    long   *iv0inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
ivo	output	Four-byte integer output vector
ivoinc	input	Output vector increment
n	input	Number of points to output

Return Value

None: void function.

Description

Converts the real input vector rv1 to integer by rounding and places the result in a four-byte integer output vector ivo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fixbnP

Vector Fix to One-byte Integer and Round

Function Definitions

```
void fixbn(
                    float         *rv1,
                    long           rv1inc,
                    unsigned char *bv0,
                    long           bv0inc,
                    long           n);

void fixbn_(
                    float         *rv1,
                    long          *rv1inc,
                    unsigned char *bv0,
                    long          *bv0inc,
                    long          *n);

void fixbnd(
                    double        *rv1,
                    long           rv1inc,
                    unsigned char *bv0,
                    long           bv0inc,
                    long           n);

void fixbnd_(
                    double        *rv1,
                    long          *rv1inc,
                    unsigned char *bv0,
                    long          *bv0inc,
                    long          *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
bvo	output	One-byte integer output vector
bvoinc	input	Output vector increment
n	input	Number of points to output

Return Value

None: void function.

Description

Converts the real input vector rv1 to integer by rounding and places the result in an onebyte integer output vector bvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

flt2P

Float Integer (2 Byte) Vector

Function Definitions

```
void flt2(
                    short *sv1,
                    long   sv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void flt2_(
                    short *sv1,
                    long  *sv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void flt2d(
                    short  *sv1,
                    long    sv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void flt2d_(
                    short  *sv1,
                    long   *sv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
sv1	input	Integer input vector
sv1inc	input	Input vector increment
rvo	output	Real vector output
rvoinc	input	Output vector increment
n	input	Number of points to float

Return Value

None: void function.

Description

Converts the input two byte integer vector sv1 to single or double precision floating point and puts the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

flt2iqP

Float Integer (2 Byte) I,Q pairs and demux 2 channels

Function Definitions

```
void flt2iq(
                    short *sv1,
                    long   sv1inc,
                    float *cvo1,
                    float *cvo2,
                    long   n);

void flt2iq_(
                    short *sv1,
                    long  *sv1inc,
                    float *cvo1,
                    float *cvo2,
                    long  *n);

void flt2iqd(
                    short  *sv1,
                    long    sv1inc,
                    double *cvo1,
                    double *cvo2,
                    long    n);

void flt2iqd_(
                    short  *sv1,
                    long   *sv1inc,
                    double *cvo1,
                    double *cvo2,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
sv1	input	Integer input vector
sv1inc	input	Input vector increment
cvo1	output	Complex output (1st I,Q channel)
cvo2	output	Complex output (2nd I,Q channel)
n	input	Number of complex outputs in cvo1

Return Value

None: void function.

Description

Special function to float two, multiplexed I,Q channels stored as 16-bit signed integer values. Output consists of two floating-point complex (I,Q) channels.

Restrictions and Notes

1. The strided elements in the input and two output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

flt2leP

Float Integer (2 Byte Little Endian) Vector

Function Definitions

```
void flt2le(
                    short *sv1,
                    long   sv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void flt2le_(
                    short *sv1,
                    long  *sv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void flt2led(
                    short  *sv1,
                    long    sv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void flt2led_(
                    short  *sv1,
                    long   *sv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
sv1	input	Integer input vector
inc1	input	Input vector increment
rvo	output	Real vector output
inco	input	Output vector increment
n	input	Number of points to float

Return Value

None: void function.

Description

Converts the input two byte integer vector sv1 to single or double precision floating point and puts the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

flt4P

Float Integer (4 Byte) Vector

Function Definitions

```
void flt4(
                    long  *iv1,
                    long   iv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void flt4_(
                    long  *iv1,
                    long  *iv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void flt4d(
                    long   *iv1,
                    long    iv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void flt4d_(
                    long   *iv1,
                    long   *iv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
iv1	input	Integer input vector
iv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points to convert

Return Value

None: void function.

Description

Converts the input four byte integer vector iv1 to single or double precision floating point and puts the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

flt4leP

Float Integer (4 Byte Little Endian) Vector

Function Definitions

```
void flt4le(
                    long  *iv1,
                    long   iv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void flt4le_(
                    long  *iv1,
                    long  *iv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void flt4led(
                    long   *iv1,
                    long    iv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void flt4led_(
                    long   *iv1,
                    long   *iv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
iv1	input	Integer input vector
iv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points to convert

Return Value

None: void function.

Description

Swap the four bytes of input vector iv1 and convert the integer to single or double precision floating point and puts the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fltbP

Float Signed Byte

Function Definitions

```
void fltb(
                    char  *bv1,
                    long   bv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void fltb_(
                    char  *bv1,
                    long  *bv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void fltbd(
                    char   *bv1,
                    long    bv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void fltbd_(
                    char   *bv1,
                    long   *bv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
bv1	input	One-byte integer input vector
bv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points to convert

Return Value

None: void function.

Description

Converts the one-byte integer input vector bv1 to single or double precision floating point and places the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fltbuP

Float Byte Unsigned

Function Definitions

```
void fltbu(
                    unsigned char *bv1,
                    long           bv1inc,
                    float         *rvo,
                    long           rvoinc,
                    long           n);

void fltbu_(
                    unsigned char *bv1,
                    long          *bv1inc,
                    float         *rvo,
                    long          *rvoinc,
                    long          *n);

void fltbud(
                    unsigned char *bv1,
                    long           bv1inc,
                    double        *rvo,
                    long           rvoinc,
                    long           n);

void fltbud_(
                    unsigned char *bv1,
                    long          *bv1inc,
                    double        *rvo,
                    long          *rvoinc,
                    long          *n);
```

Parameters

Argument	I/O	Description
bv1	input	One-byte integer input vector
bv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points to convert

Return Value

None: void function.

Description

Converts the one-byte unsigned integer input vector bv1 to single or double precision floating point and places the result in vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fxsl2nP

Vector Scale, Limit, Fix to Two-byte Integer and Round

Function Definitions

```
void fxsl2n(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    short *sv0,
                    long   sv0inc,
                    long   n);

void fxsl2n_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    short *sv0,
                    long  *sv0inc,
                    long  *n);

void fxsl2nd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    short  *sv0,
                    long    sv0inc,
                    long    n);

void fxsl2nd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    short  *sv0,
                    long   *sv0inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	input	Real scalar multiplier
rs2	input	Real scalar adder
svo	output	Two-byte integer output vector
svoinc	input	Output vector increment
n	input	Number of points to output

Return Value

None: void function.

Description

Converts the real input vector rv1 by multiplying by rs1, adding rs2, limiting to (-2**15, 2**15-1) and rounding to nearest integer value. The result is placed in a two-byte output integer vector svo, for n points.

Restrictions and Notes

1. The strided elements in the two input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fxsl4nP

Vector Scale, Limit, Fix to Four-byte Integer and Round

Function Definitions

```
void fxsl4n(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    long  *iv0,
                    long   iv0inc,
                    long   n);

void fxsl4n_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    long  *iv0,
                    long  *iv0inc,
                    long  *n);

void fxsl4nd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    long   *iv0,
                    long    iv0inc,
                    long    n);

void fxsl4nd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    long   *iv0,
                    long   *iv0inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	input	Real scalar multiplier
rs2	input	Real scalar adder
ivo	output	Four-byte integer output vector
ivoinc	input	Output vector increment
n	input	Number of points to output

Return Value

None: void function.

Description

Converts the real input vector rv1 by multiplying by rs1, adding rs2, limiting that to between -2^31 and 2^31-1, and rounding to nearest integer value. The result is placed in a four-byte integer output vector ivo, for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

fxslbnP

Vector Scale, Limit, Fix to One-byte Integer and Round

Function Definitions

```
void fxslbn(
                    float         *rv1,
                    long           rv1inc,
                    float         *rs1,
                    float         *rs2,
                    unsigned char *bv0,
                    long           bv0inc,
                    long           n);

void fxslbn_(
                    float         *rv1,
                    long          *rv1inc,
                    float         *rs1,
                    float         *rs2,
                    unsigned char *bv0,
                    long          *bv0inc,
                    long          *n);

void fxslbnd(
                    double        *rv1,
                    long           rv1inc,
                    double        *rs1,
                    double        *rs2,
                    unsigned char *bv0,
                    long           bv0inc,
                    long           n);

void fxslbnd_(
                    double        *rv1,
                    long          *rv1inc,
                    double        *rs1,
                    double        *rs2,
                    unsigned char *bv0,
                    long          *bv0inc,
                    long          *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	input	Real scalar multiplier
rs2	input	Real scalar adder
bvo	output	One-byte integer output vector
bvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Converts the real input vector rv1 by multiplying by rs1, adding rs2, limiting between 0 and 255, and rounding to nearest integer value. The result is placed in the one-byte integer output vector bvo, for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

gaussian

Gaussian Window

Function Definitions

```
void gaussian(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    float  alpha);

void gaussian(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    float *alpha);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process
alpha	input	Alpha parameter for window computation

Return Value

None: void function.

Description

This function generates a Gaussian window, multiplying the window by the real input vector rv1 to produce the output vector rv2.

Restrictions and Notes

1. The strided elements in the input and output buffers must be than **n**.
2. The operation is performed faster if the **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

gcexpP

Generate a Complex Exponential with Constant Rotation

Function Definitions

```
void gcexp(
                    float *cs1,
                    float *cs2,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void gcexp_(
                    float *cs1,
                    float *cs2,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void gcexpd(
                    double *cs1,
                    double *cs2,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void gcexpd_(
                    double *cs1,
                    double *cs2,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cs1	input	Complex input scalar (phase step)
cs2	input	Complex input scalar (initial phase)
cvo	output	Complex output vector
cvoinc	input	Sample increment of cvo
n	input	Number of samples to output

Return Value

None: void function.

Description

Generates a complex exponential consisting of a unit vector rotating at a constant angular frequency. Each complex output point contains the cosine and sine of the rotation angle in the real and imaginary parts, respectively. Input complex scalar cs1 contains the cosine and sine of the angular increment between output points. Input complex scalar cs2 contains the cosine and sine of the starting phase angle. For maximum execution speed, each output point is computed from the complex multiplication of the previous point by the complex phase step. For very long vectors (n>10,000) this may introduce cumulative roundoff errors when the phase step is large.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

gcosfP

Vector Generate Cosines

Function Definitions

```
void gcosf(
                    float *rvo,
                    long   rvoinc,
                    float *rs1,
                    float *rs2,
                    long   n);

void gcosf_(
                    float *rvo,
                    long  *rvoinc,
                    float *rs1,
                    float *rs2,
                    long  *n);

void gcosfd(
                    double *rvo,
                    long    rvoinc,
                    double *rs1,
                    double *rs2,
                    long    n);

void gcosfd_(
                    double *rvo,
                    long   *rvoinc,
                    double *rs1,
                    double *rs2,
                    long   *n);
```

Parameters

Argument	I/O	Description
rvo	output	Real output vector
rvoinc	input	Output vector increment
rs1	input	Real scalar frequency
rs2	input	Real scalar initial phase
n	input	Number of points in circle

Return Value

None: void function.

Description

Output vector rvo is filled with cosine values computed from the initial phase rs2 and frequency rs1, for n points. rs1 and rs2 are specified in units of fraction of a circle.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

hammP

Apply a Hamming Window to a Real Vector

Function Definitions

```
void hamm(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void hamm_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void hammd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void hammd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies each element of the input vector rv1 by the Hamming window.

Restrictions and Notes

1. The strided elements in the input and output buffers must be greater than **n**.
2. The operation is performed faster if the **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

hannP

Apply a Hanning Window to a Real Vector

Function Definitions

```
void hann(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    long   flg);

void hann_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    long  *flg);

void hannd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    long    flg);

void hannd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    long   *flg);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies each element of the input vector rv1 by the Hanning window.

Restrictions and Notes

1. The strided elements in the input and output buffers must be greater than **n**.
2. The operation is performed faster if the **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

hlbrtP

Hilbert Transform

Function Definitions

```
void hlbrt(
                    float *rv1,
                    float *rvo,
                    long   n);

void hlbrt_(
                    float *rv1,
                    float *rvo,
                    long  *n);

void hlbrtd(
                    double *rv1,
                    double *rvo,
                    long    n);

void hlbrtd_(
                    double *rv1,
                    double *rvo,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rvo	output	Real output vector
n	input	transform size (number of real points)

Return Value

None: void function.

Description

The Hilbert transform of the input vector is computed and stored in the output vector. The transform is computed from a forward real-to-complex FFT, a complex multiplication by -j, followed by an inverse complex-to-real FFT.

Restrictions and Notes

1. The input and output buffers must be of type compact.
2. The input and output buffers must be greater than **n** in length.
3. The operation is performed faster if the **n** is a power of two.
4. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.

imgfirP

Filter MxN Convolution

Function Definitions

```
void imgfir(
                    float *rv1,
                    long   m,
                    long   n,
                    float *rv2,
                    float *rvo,
                    long   p,
                    long   q);

void imgfir_(
                    float *rv1,
                    long  *m,
                    long  *n,
                    float *rv2,
                    float *rvo,
                    long  *p,
                    long  *q);

void imgfird(
                    double *rv1,
                    long    m,
                    long    n,
                    double *rv2,
                    double *rvo,
                    long    p,
                    long    q);

void imgfird_(
                    double *rv1,
                    long   *m,
                    long   *n,
                    double *rv2,
                    double *rvo,
                    long   *p,
                    long   *q);
```

Parameters

Argument	I/O	Description
rv1	input	Input image
m	input	Number of rows in input and output
n	input	Number of columns in input and output
rv2	input	pxq input matrix of coefficients
rvo	output	Output image
p	input	Number of rows in rv2
q	input	Number of columns in rv2

Return Value

None: void function.

Description

Filter an m-by-n image by performing a two-dimensional convolution with a p-by-q kernel. The perimeter of the output image is padded with a border of (p-1)/2 rows of zeros on top and bottom and (q-1)/2 columns of zeros on left and right.

Restrictions and Notes

kaiser

Kaiser Window

Function Definitions

```
void kaiser(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    float  alpha);

void kaiser_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    float *alpha);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process
alpha	input	Alpha parameter for window computation

Return Value

None: void function.

Description

This function generates a Kaiser window, multiplying the window by the real input vector rv1 to produce the output vector rv2.

Restrictions and Notes

1. The strided elements in the input and output buffers must be greater than **n**.
2. The operation is performed faster if the **n** is a power of two.
3. The operation is performed faster if the input and output buffers are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

lveqP

Logical Vector Equal

Function Definitions

```
void lveq(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lveq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lveqd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lveqd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vector points: if point in input vector rv1 is equal to point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvgeP

Logical Vector Greater Than or Equal

Function Definitions

```
void lvge(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvge_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvged(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvged_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	output	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vector points: if point in input vector rv1 is greater than or equal to point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvgtP

Logical Vector Greater

Function Definitions

```
void lvgt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvgt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvgtd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvgtd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	output	Vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vector points: if point in input vector rv1 is greater than the point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvleP

Logical Vector Less Than or Equal

Function Definitions

```
void lvle(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvle_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvled(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvled_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	output	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vector points: if point in input vector rv1 is less than or equal to point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvieq

Integer Vector Equal

Function Definitions

```
void lvieq(
                    long          *iv1,
                    long           iv1inc,
                    long          *iv2,
                    long           iv2inc,
                    long          *iv0,
                    long           iv0inc,
                    unsigned long  n);

void lvieq_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iv2,
                    long          *iv2inc,
                    long          *iv0,
                    long          *iv0inc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rv2	input	starting address of long integer vector 2
rv2inc	input	increment for vector 2
rvo	output	output long integer vector
rvoinc	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2, setting rvo according to the result. The result is 0xffff_ffff (i.e., -1) if the values are equal. Otherwise the resulting value is 0.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

lvigt

Integer Vector Greater

Function Definitions

```
void lvigt(
                    long          *iv1,
                    long           iv1inc,
                    long          *iv2,
                    long           iv2inc,
                    long          *iv0,
                    long           iv0inc,
                    unsigned long  n);

void lvigt_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iv2,
                    long          *iv2inc,
                    long          *iv0,
                    long          *iv0inc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rv2	input	starting address of long integer vector 2
rv2inc	input	increment for vector 2
rvo	output	output long integer vector
rvoinc	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2, setting rvo according to the result. The result is 0xffff_ffff (i.e., -1) if $rv1[i] > rv2[i]$. Otherwise the resulting value is 0.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

lvilt

Integer Vector Less Than

Function Definitions

```
void lvilt(
                    long          *iv1,
                    long           iv1inc,
                    long          *iv2,
                    long           iv2inc,
                    long          *iv0,
                    long           iv0inc,
                    unsigned long  n);

void lvilt_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iv2,
                    long          *iv2inc,
                    long          *iv0,
                    long          *iv0inc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rv2	input	starting address of long integer vector 2
rv2inc	input	increment for vector 2
rvo	output	output long integer vector
rvoinc	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2, setting rvo according to the result. The result is 0xffff_ffff (i.e., -1) if $rv1[i] < rv2[i]$. Otherwise the resulting value is 0

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

lvine

Integer Vector Not Equal

Function Definitions

```
void lvine(
                    long          *iv1,
                    long           iv1inc,
                    long          *iv2,
                    long           iv2inc,
                    long          *iv0,
                    long           iv0inc,
                    unsigned long  n);

void lvine_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iv2,
                    long          *iv2inc,
                    long          *iv0,
                    long          *iv0inc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rv2	input	starting address of long integer vector 2
rv2inc	input	increment for vector 2
rvo	output	output long integer vector
rvoinc	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2, setting rvo according to the result. The result is 0xffff_ffff (i.e., -1) if the values are not equal. Otherwise the resulting value is 0

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

lvinot

Integer Complement

Function Definitions

```
void lvinot(
                    long          *iv1,
                    long           iv1inc,
                    long          *iv0,
                    long           iv0inc,
                    unsigned long  n);

void lvinot_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iv0,
                    long          *iv0inc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rvo	output	output long integer vector
rvoinc	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function computes the one's complement of each element in the input vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

lvltP

Logical Vector Less

Function Definitions

```
void lvlt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvlt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvltd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvltd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vectors: if point in input vector rv1 is less than the point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvneP

Logical Vector Not Equal

Function Definitions

```
void lvne(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvne_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvned(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvned_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in two input vectors: if point on input vector rv1 is not equal to point in input vector rv2, put 1.0 into output vector rvo; else put 0.0 into output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

lvnotP

Logical Vector Not Equal to Zero

Function Definitions

```
void lvnot(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void lvnot_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void lvnotd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void lvnotd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare n points in input vector to zero: if point in input vector rv1 is equal to zero, put 1.0 into output vector rvo; else put 0.0 into output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

maxmgvP

Maximum Magnitude Element of Vector

Function Definitions

```
void maxmgv(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long  *idx,
                    long   n);

void maxmgv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *idx,
                    long  *n);

void maxmgvd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long   *idx,
                    long    n);

void maxmgvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Maximum magnitude output
idx	input	Index of maximum magnitude
n	input	Number of points to process

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the greatest magnitude. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the input vector is fully aligned for the target hardware.
3. This operation is supported in both single and double precision.
4. Complex buffers are in Complex Interleaved format.

maxvP

Find the Maximum Value and its Location

Function Definitions

```
void maxv(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long  *idx,
                    long   n);

void maxv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *idx,
                    long  *n);

void maxvd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long   *idx,
                    long    n);

void maxvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rso	output	Real scalar output
idx	input	Index of maximum point
n	input	Number of points to process

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the highest value. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the input vector is fully aligned for the target hardware.
3. This operation is supported in both single and double precision.
4. Complex buffers are in Complex Interleaved format.

maxvi

Integer Max Index

Function Definitions

```
void maxvi(
                    long          *iv1,
                    long           iv1inc,
                    long          *iso,
                    long          *idx,
                    unsigned long  n);

void maxvi_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iso,
                    long          *idx,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
iv1	input	pointer to input vector
iv1inc	input	Input vector increment
is0	output	pointer to max value
idx	output	resulting index
n	input	number of elements to process

Return Value

None: void function.

Description

This function scans the input vector and locates the element having the maximum value. This function stores the maximum value at *iso, and stores the index of the value at *idx. The index is incremented by iv1inc, so that the maximum value actually occurs at: $iv1[(*idx - 1) * iv1inc]$.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

meamgvP

Mean Magnitude of Real Vector

Function Definitions

```
void meamgv(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void meamgv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void meamgvd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void meamgvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points to process

Return Value

None: void function.

Description

Mean value of absolute value of elements of rv1 is returned in real scalar rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

meanc

Complex Mean of Vector Elements

Function Definitions

```
void meanc(
                    float *cv1,
                    long   cv1inc,
                    float *cso,
                    long   n);

void meanc_(
                    float *cv1,
                    long  *cv1inc,
                    float *cso,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
cso	output	Where to deposit complex mean
n	input	Number of samples to process

Return Value

None: void function.

Description

This function sums all complex elements in the vector cv1, and then divides by n, storing the complex result at *cs1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. Complex buffers are in Complex Interleaved format.

meanvP

Mean of Real Vector

Function Definitions

```
void meanv(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void meanv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void meanvd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void meanvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points to process

Return Value

None: void function.

Description

Mean value of elements in rv1 is returned in real scalar rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

measqvP

Mean of Square of Real Vector

Function Definitions

```
void measqv(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void measqv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void measqvd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void measqvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points to process

Return Value

None: void function.

Description

Mean value of square of elements of rv1 is returned in real scalar rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

minmgvP

Find the Minimum Magnitude and its Location

Function Definitions

```
void minmgv(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void minmgv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void minmgvd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void minmgvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rso	output	Real scalar output
idx	input	Position in vector
n	input	Number of points to process

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the least magnitude. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

minvP

Find the Minimum Value and its Position

Function Definitions

```
void minv(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void minv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void minvd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void minvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rso	output	Real scalar output
idx	input	Position of minimum
n	input	Number of points in output vector

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the least value. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 1.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

minvi

Integer Min Index

Function Definitions

```
void minvi(
                    long          *iv1,
                    long           iv1inc,
                    long          *iso,
                    long          *idx,
                    unsigned long  n);

void minvi_(
                    long          *iv1,
                    long          *iv1inc,
                    long          *iso,
                    long          *idx,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
iv1	input	pointer to input vector
iv1inc	input	Input vector increment
iso	output	pointer to max value
idx	output	resulting index
n	input	number of elements to process

Return Value

None: void function.

Description

This function scans the input vector and locates the element having the minimum value. This function stores the minimum value at *iso, and stores the index of the value at *idx. The index is incremented by iv1inc, so that the minimum value actually occurs at: $iv1[(*idx - 1) * iv1inc]$.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

mom2v

2nd Moment of a Vector

Function Definitions

```
void mom2v(
                    float *rv1,
                    long   rv1inc,
                    float  c,
                    float *rs1,
                    long   n);

void mom2v_(
                    float *rv1,
                    long  *rv1inc,
                    float *c,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
c	input	center of mass for moment computation
rs1	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the second moment about the point **c**. For each point in the vector (i), the square of the difference between $\mathbf{rv1}[\mathbf{i}]$ and **c** is accumulated. The result is the average of this value.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

mom3v

3rd Moment of a Vector

Function Definitions

```
void mom3v(
                    float *rv1,
                    long   rv1inc,
                    float  c,
                    float *rs1,
                    long   n);

void mom3v_(
                    float *rv1,
                    long  *rv1inc,
                    float *c,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
c	input	center of mass for moment computation
rs1	output	Where to deposit real result
n	input	Number of samples to proces

Return Value

None: void function.

Description

This function computes the third moment about the point **c**. For each point in the vector (i), the cube of the difference between $\mathbf{rv1}[\mathbf{i}]$ and **c** is accumulated. The result is the average of this value.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

mom4v

4th Moment of a Vector

Function Definitions

```
void mom4v(
                    float *rv1,
                    long   rv1inc,
                    float  c,
                    float *rs1,
                    long   n);

void mom4v_(
                    float *rv1,
                    long  *rv1inc,
                    float *c,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
c	input	center of mass for moment computation
rs1	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the third moment about the point **c**. For each point in the vector (i), the 4th power of the difference between $\mathbf{rv1}[\mathbf{i}]$ and **c** is accumulated. The result is the average of this value.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

mtranspose

Real Matrix Transpose

Function Definitions

```
int mtranspose(
                    float *rm1,
                    long   nr,
                    long   nc,
                    long   rm1stride,
                    float *rmo,
                    long   rmostride);

int mtranspose_(
                    float *rm1,
                    long  *nr,
                    long  *nc,
                    long  *rm1stride,
                    float *rmo,
                    long  *rmostride);
```

Parameters

Argument	I/O	Description
rm1	input	input real matrix
nr	input	number of rows in the input amd output matrix
nc	input	number of columns in the input and output matrix
rm1stride	input	row stride of the input matrix
rmo	output	output real matrix
rmostride	input	output row stride

Return Value

The mtranspose function returns an integer $<= 0$:

- 0 indicates success;
- < 0 indicates invalid parameters.

Description

This function transposes an input matrix or sub-matrix. The input matrix has dimensions **nr** by **nc**. The input row stride is the distance, in samples, from one row to the next. Normally, the stride is the same as **nc**. However, a sub-matrix can be transposed by specifying the actual number of columns in the input vector as **cm1stride** and the number of columns in the submatrix as **nc**. Note that **nc** is always $<=$ **cm1stride**. The resulting matrix is deposited at **cmo**, and has dimensions **nc** by **nr**. Note that the output stride is normally equal to nr. However, if the result is a submatrix of a larger matrix, then orstride is the number of columns in the output matrix, and nr is the number of columns in the transposed submatrix. If the matrix is square, the function can be performed in-place. Otherwise, **cm1** must not equal **cmo**.

Restrictions and Notes

1. The function will return a non-zero value if an error occurs.
2. The operation is performed faster when **nr** and **nc** are of power of two.
3. The operation is performed faster when **rm1stride** and **rmostride** are of power of two.
4. The rows in each matrix (series of vectors) are of type compact.
5. When nc equals cm1stride then the whole input matrix is of type compact.
6. When nc equals cmostride then the whole output matrix is of type compact.
7. The operation is performed faster when the input and output matrix are fully aligned for the target hardware.
8. This operation is only supported in single precision.
9. An in-place operation can only be performed when the matrix is square.

mvessqP

Mean of Signed Square of Vector Elements

Function Definitions

```
void mvessq(
                    float *rv1,
                    float *rso,
                    long   n);

void mvessq_(
                    float *rv1,
                    float *rso,
                    long  *n);

void mvessqd(
                    double *rv1,
                    double *rso,
                    long    n);

void mvessqd_(
                    double *rv1,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rso	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

Mean value of signed square of elements of rv1 is returned in real scalar rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

mxcholesky

Cholesky Decomposition

Function Definitions

```
int mxcholesky(
                    float *rm1,
                    long   rm1rinc,
                    long   rm1cinc,
                    float *rmo,
                    long   rmorinc,
                    long   rmocinc,
                    long   nrc);

int mxcholesky_(
                    float *rm1,
                    long  *rm1rinc,
                    long  *rm1cinc,
                    float *rmo,
                    long  *rmorinc,
                    long  *rmocinc,
                    long  *nrc);
```

Parameters

Argument	I/O	Description
rm1	input	Input real matrix
rm1cinc	input	row to row stride
rm1rinc	input	column to column stride
rco	output	Real output matrix
rmorinc	input	row to row stride
rmocinc	input	column to column stride
ncr	input	number of rows = number of columns

Return Value

None: void function.

Description

This function computes the Cholesky decomposition of the real square symmetric matrix min. The matrix is nrc rows by nrc columns. The matrix must be positive definite. If the input matrix is not positive definite, then the Cholesky decomposition does not exist, and the function returns a nonzero value. The Cholesky decomposition is returned in the matrix mout. This matrix is real square nrc x nrc, but the result resides within the lower triangle and the diagonal. Entries above the diagonal are not part of the solution. This function cannot be performed in-place. For more information on Cholesky, see Numerical Recipes in C, 2nd Edition, Press, Teukolsky, Vetterling, and Flannery.

Restrictions and Notes

1. The operation is performed faster when **ncr** is of power of two.
2. The operation is performed faster when **rm1rinc** and **rmorinc** are of power of two.
3. The operation is performed faster when the input and output matrix are fully aligned for the target hardware.
4. This operation is only supported in single precision.

mxchsolve

Cholesky Solve

Function Definitions

```
int mxchsolve(
                    float *rvy,
                    long   rvyinc,
                    float *rmM,
                    long   rmMrinc,
                    long   rmMcinc,
                    float *rvx,
                    long   rvxinc,
                    long   nrc);

int mxchsolve_(
                    float *rvy,
                    long  *rvyinc,
                    float *rmM,
                    long  *rmMrinc,
                    long  *rmMcinc,
                    float *rvx,
                    long  *rvxinc,
                    long  *nrc);
```

Parameters

Argument	I/O	Description
rvy	input	Input real vector y
rvyinc	input	Input vector y stride
rmM	input	Input Cholesky decomposition
rmMrinc	input	row to row stride
rmMcinc	input	column to column stride
rvx	output	Output real vector x
rvxinc	input	vector x stride
ncr	input	number of rows = number of columns

Return Value

None: void function.

Description

This function computes solves the linear set of equations $\mathrm{xM} = \mathrm{y}$ for a matrix **M** and a real vector **y**. **cmM** is the Cholesky decomposition of the real matrix M (see âĂIJmxcholesky Cholesky DecompositionâĂİ on page 2-166). This function cannot be performed in-place. For more information on Cholesky, see Numerical Recipes in C, 2nd Edition, Press, Teukolsky, Vetterling, and Flannery.

Restrictions and Notes

1. The operation is performed faster when **ncr** is of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is only supported in single precision.

mxmlaP

Real Matrix Multiply and Add

Function Definitions

```
int mxmla(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rm2,
                    long   rm2cinc,
                    long   rm2rinc,
                    float *rmo,
                    long   rmocinc,
                    long   rmorinc,
                    long   nro,
                    long   nco,
                    long   ncr);

int mxmla_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rm2,
                    long  *rm2cinc,
                    long  *rm2rinc,
                    float *rmo,
                    long  *rmocinc,
                    long  *rmorinc,
                    long  *nro,
                    long  *nco,
                    long  *ncr);

int mxmlad(
                    double *rm1,
                    long    rm1cinc,
                    long    rm1rinc,
                    double *rm2,
                    long    rm2cinc,
                    long    rm2rinc,
                    double *rmo,
                    long    rmocinc,
                    long    rmorinc,
                    long    nro,
                    long    nco,
                    long    ncr);
```

```
int mxmlad_(
                double *rm1,
                long   *rm1cinc,
                long   *rm1rinc,
                double *rm2,
                long   *rm2cinc,
                long   *rm2rinc,
                double *rmo,
                long   *rmocinc,
                long   *rmorinc,
                long   *nro,
                long   *nco,
                long   *ncr);
```

Parameters

Argument	I/O	Description
rm1	input	First real matrix
rm1cinc	input	Sample increment between columns
rm1rinc	input	Sample increment between rows
rm2	input	Second real matrix to move
rm2cinc	input	Sample increment between columns
rm2rinc	input	Sample increment between rows
rmo	output	Real output matrix
rmocinc	input	Sample increment between columns
rmorinc	input	Sample increment between rows
nro	input	Number of rows in output matrix
nco	input	Number of columns in output matrix
ncr	input	Number of columns in rm1 and rows in rm2

Return Value

None: void function.

Description

Performs a real matrix multiply between **rm1** and **rm2**. The result is added to **rmo**.

Restrictions and Notes

1. The operation is performed faster when bf nro, **nco** and **ncr** are of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is supported in single and double precision.

mxmlsP

Real Matrix Multiply and Subtract

Function Definitions

```
int mxmls(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rm2,
                    long   rm2cinc,
                    long   rm2rinc,
                    float *rmo,
                    long   rmocinc,
                    long   rmorinc,
                    long   nro,
                    long   nco,
                    long   ncr);

int mxmls_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rm2,
                    long  *rm2cinc,
                    long  *rm2rinc,
                    float *rmo,
                    long  *rmocinc,
                    long  *rmorinc,
                    long  *nro,
                    long  *nco,
                    long  *ncr);

int mxmlsd(
                    double *rm1,
                    long    rm1cinc,
                    long    rm1rinc,
                    double *rm2,
                    long    rm2cinc,
                    long    rm2rinc,
                    double *rmo,
                    long    rmocinc,
                    long    rmorinc,
                    long    nro,
                    long    nco,
                    long    ncr);
```

```
int mxmlsd_(
                double *rm1,
                long   *rm1cinc,
                long   *rm1rinc,
                double *rm2,
                long   *rm2cinc,
                long   *rm2rinc,
                double *rmo,
                long   *rmocinc,
                long   *rmorinc,
                long   *nro,
                long   *nco,
                long   *ncr);
```

Parameters

Argument	I/O	Description
rm1	input	First real matrix
rm1cinc	input	Sample increment between columns
rm1rinc	input	Sample increment between rows
rm2	input	Second real matrix to move
rm2cinc	input	Sample increment between columns
rm2rinc	input	Sample increment between rows
rmo	output	Real output matrix
rmocinc	input	Sample increment between columns
rmorinc	input	Sample increment between rows
nro	input	Number of rows in output matrix
nco	input	Number of columns in output matrix
ncr	input	Number of columns in rm1 and rows in rm2

Return Value

None: void function.

Description

Performs a real matrix multiply between **rm1** and **rm2**. The result is subtracted from **rmo**.

Restrictions and Notes

1. The operation is performed faster when bf nro, **nco** and **ncr** are of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is supported in single and double precision.

mxmovP

Real Matrix / Submatrix move

Function Definitions

```
int mxmov(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rmo,
                    long   rmocinc,
                    long   rmorinc,
                    long   nro,
                    long   nco);

int mxmov_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rmo,
                    long  *rmocinc,
                    long  *rmorinc,
                    long  *nro,
                    long  *nco);

int mxmovd(
                    double *rm1,
                    long    rm1cinc,
                    long    rm1rinc,
                    double *rmo,
                    long    rmocinc,
                    long    rmorinc,
                    long    nro,
                    long    nco);

int mxmovd_(
                    double *rm1,
                    long   *rm1cinc,
                    long   *rm1rinc,
                    double *rmo,
                    long   *rmocinc,
                    long   *rmorinc,
                    long   *nro,
                    long   *nco);
```

Parameters

Argument	I/O	Description
rm1	input	Real matrix to move
rm1cinc	input	Sample increment along columns
rm1rinc	input	Sample increment along rows
rmo	output	Real output matrix
rmocinc	input	Sample increment along columns
rmorinc	input	Sample increment along rows
nr	input	Number of rows
nc	input	Number of columns

Return Value

None: void function.

Description

Copies the desired points in one matrix to the second matrix.

Restrictions and Notes

1. The operation is performed faster when bf nro and **nco** are of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is supported in single and double precision.

mxmulP

Matrix Multiply

Function Definitions

```
int mxmul(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rm2,
                    long   rm2cinc,
                    long   rm2rinc,
                    float *rmo,
                    long   rmocinc,
                    long   rmorinc,
                    long   nro,
                    long   nco,
                    long   ncr);

int mxmul_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rm2,
                    long  *rm2cinc,
                    long  *rm2rinc,
                    float *rmo,
                    long  *rmocinc,
                    long  *rmorinc,
                    long  *nro,
                    long  *nco,
                    long  *ncr);

int mxmuld(
                    double *rm1,
                    long    rm1cinc,
                    long    rm1rinc,
                    double *rm2,
                    long    rm2cinc,
                    long    rm2rinc,
                    double *rmo,
                    long    rmocinc,
                    long    rmorinc,
                    long    nro,
                    long    nco,
                    long    ncr);
```

```
int mxmuld_(
                double *rm1,
                long   *rm1cinc,
                long   *rm1rinc,
                double *rm2,
                long   *rm2cinc,
                long   *rm2rinc,
                double *rmo,
                long   *rmocinc,
                long   *rmorinc,
                long   *nro,
                long   *nco,
                long   *ncr);
```

Parameters

Argument	I/O	Description
rm1	input	First real matrix
rm1cinc	input	Sample increment along columns
rm1rinc	input	Sample increment along rows
rm2	input	Second real matrix to move
rm2cinc	input	Sample increment along columns
rm2rinc	input	Sample increment along rows
rmo	output	Real output matrix
rmocinc	input	Sample increment along columns
rmorinc	input	Sample increment along rows
nro	input	Number of rows in output matrix
nco	input	Number of columns in output matrix
ncr	input	Number of columns in rm1 and rows in rm2

Return Value

None: void function.

Description

Performs a real matrix multiply.

Restrictions and Notes

1. The operation is performed faster when bf nro, **nco** and **ncr** are of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is supported in single and double precision.

mxsma

Real Scalar Matrix Multiply and Add

Function Definitions

```
int mxsma(
                    float *rmx,
                    long   rmxcinc,
                    long   rmxrinc,
                    float *rsa,
                    float *rmy,
                    long   rmycinc,
                    long   rmyrinc,
                    float *rmo,
                    long   rmzcinc,
                    long   rmzrinc,
                    long   nro,
                    long   nco);

int mxsma_(
                    float *rmx,
                    long  *rmxcinc,
                    long  *rmxrinc,
                    float *rsa,
                    float *rmy,
                    long  *rmycinc,
                    long  *rmyrinc,
                    float *rmo,
                    long  *rmzcinc,
                    long  *rmzrinc,
                    long  *nro,
                    long  *nco);
```

Parameters

Argument	I/O	Description
rmx	input	Input real matrix
rmxrinc	input	row to row stride
rmxcinc	input	column to column stride
rsa	input	real scalar
rmy	input	Input real matrix
rmyrinc	input	row to row stride
rmycinc	input	column to column stride
rmz	output	Output real matrix
rmzrinc	input	row to row stride
rmzcinc	input	column to column stride
nr	input	number of rows
nc	input	number of cols

Return Value

None: void function.

Description

This function computes $\mathbf{Z} = \mathbf{aX} + \mathbf{Y}$ for the real scalar a and real matrices **Z**, **X**, and **Y**.

Restrictions and Notes

1. The operation is performed faster when bf nro and **nco** are of power of two.
2. The operation is performed faster when the input and output buffers are fully aligned for the target hardware.
3. This operation is only supported in single precision.

mxstddev

Standard Deviation of Real Matrix

Function Definitions

```
int mxstddev(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rs2,
                    long   nr,
                    long   nc);

int mxsma_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rs2,
                    long  *nr,
                    long  *nc);
```

Parameters

Argument	I/O	Description
rm1	input	Input real matrix
rm1cinc	input	distance from one row to the next
rm1rinc	input	distance from one column to the next
rs2	output	Where to deposit real result
nr	input	Number of rows in input matrix
nc	input	Number of columns in input matrix

Return Value

None: void function.

Description

This function computes the standard deviation of all elements in the input matrix. The standard deviation is the square root of the variance.

Restrictions and Notes

1. The operation is performed faster when bf nr and **nc** are of power of two.
2. The operation is performed faster when the input buffer is fully aligned for the target hardware.
3. This operation is only supported in single precision.

mxvariance

Variance of Real Matrix

Function Definitions

```
int mxvariance(
                    float *rm1,
                    long   rm1cinc,
                    long   rm1rinc,
                    float *rs2,
                    long   nr,
                    long   nc);

int mxvariance_(
                    float *rm1,
                    long  *rm1cinc,
                    long  *rm1rinc,
                    float *rs2,
                    long  *nr,
                    long  *nc);
```

Parameters

Argument	I/O	Description
rm1	input	Input real matrix
rm1cinc	input	distance from one row to the next
rm1rinc	input	distance from one column to the next
rs2	output	Where to deposit real result
nr	input	Number of rows in input matrix
nc	input	Number of columns in input matrix

Return Value

None: void function.

Description

This function computes the variance of the input matrix. This function first computes the mean of all elements in the matrix. Then, for each point in the matrix, the square of the difference between each matrix element and the mean is accumulated. The result is the average of this value.

Restrictions and Notes

1. The operation is performed faster when bf nr and **nc** are of power of two.
2. The operation is performed faster when the input buffer is fully aligned for the target hardware.
3. This operation is only supported in single precision.

nasCW_freewts

Free FFT Complex and Real Exponential Tables for FFT Routines

Function Definitions

```
void nasCW_freewts(
                    float *p);

void nasCW_freewts_(
                    float *p);
```

Parameters

Argument	I/O	Description
p	input	Pointer to a weight used by FFT.

Return Value

None: void function.

Description

A lookup table of complex numbers is computed for performing FFTs consisting of up to m complex samples. This function is called to free up the weights and internal FFT structures.

Restrictions and Notes

nzcrosP

Find Location of Selected Zero Crossing

Function Definitions

```
void nzcros(
                    float *rv1,
                    long   rv1inc,
                    long   m,
                    long  *idx,
                    long  *nzc,
                    long   n);

void nzcros_(
                    float *rv1,
                    long  *rv1inc,
                    long  *m,
                    long  *idx,
                    long  *nzc,
                    long  *n);

void nzcrosd(
                    double *rv1,
                    long    rv1inc,
                    long    m,
                    long   *idx,
                    long   *nzc,
                    long    n);

void nzcrosd_(
                    double *rv1,
                    long   *rv1inc,
                    long   *m,
                    long   *idx,
                    long   *nzc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
m	input	Integer scalar input - desired zero crossing
idx	output	Vector index of desired zero crossing
nzc	output	Number of zero crossings found
n	input	Number of points in input vector

Return Value

None: void function.

Description

The vector **rv1** is searched for zero crossings, instances where a negative element is preceded or followed by a positive element. The position of the mth crossing is returned in **idx**, and the total number of crossings found is returned in **nzc**. The position is the index within the examined elements only and is dependent on the increment used. The first index is 0. If at least **m** crossings are not found, 0 is returned in **idx**.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.

3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. The vector idx is returned in compact form with nzc set to its length.

polarP

Rectangular to Polar Coordinate Conversion

Function Definitions

```
void polar(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void polar_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void polard(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void polard_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

The polar coordinates of each rectangular input pair are determined and written to the output vector cvo. For cvo, the real part is the radius and the imaginary part is the angle in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

rectP

Polar to Rectangular Coordinate Conversion

Function Definitions

```
void rect(
                    float *cv1,
                    long   cv1inc,
                    float *cvo,
                    long   cvoinc,
                    long   n);

void rect_(
                    float *cv1,
                    long  *cv1inc,
                    float *cvo,
                    long  *cvoinc,
                    long  *n);

void rectd(
                    double *cv1,
                    long    cv1inc,
                    double *cvo,
                    long    cvoinc,
                    long    n);

void rectd_(
                    double *cv1,
                    long   *cv1inc,
                    double *cvo,
                    long   *cvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

The rectangular coordinates are calculated from the polar coordinate pair in complex input vector cv1 and stored in complex vector cvo. For the input vector cv1, the real part is the radius and the imaginary part is the angle in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

reqsP

Find Location of First Element Equal to a Scalar

Function Definitions

```
void reqs(
                    float *rv1,
                    long   rv1inc,
                    float *rs2,
                    long   n,
                    long  *idx);

void reqs_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs2,
                    long  *n,
                    long  *idx);

void reqsd(
                    double *rv1,
                    long    rv1inc,
                    double *rs2,
                    long    n,
                    long   *idx);

void reqsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs2,
                    long   *n,
                    long   *idx);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs2	input	Real scalar input
n	input	Number of points in output vector
idx	output	Position of scalar found in rv1

Return Value

None: void function.

Description

The vector **rv1** is searched from the beginning for an element equal to the scalar **rs2**. The position of the found element is returned in **idx**, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0. If no element is found meeting the criteria, -1 is returned for **idx**.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rfftP

Real FFT, in Place - Fwd and Inv

Function Definitions

```
void rfft(
                    float *cvio,
                    long   n,
                    long   flg);

void rfft_(
                    float *cvio,
                    long  *n,
                    long  *flg);

void rfftd(
                    double *cvio,
                    long    n,
                    long    flg);

void rfftd_(
                    double *cvio,
                    long   *n,
                    long   *flg);
```

Parameters

Argument	I/O	Description
cvio	input	Real/complex input vector
n	input	FFT size (number of real points)
flg	input	Flag selecting transform direction +1 => forward, -1 => inverse

Return Value

None: void function.

Description

The forward real-to-complex or inverse complex-to-real FFT of the input vector is computed in place.

Restrictions and Notes

1. This operation is real to complex if the flag is set to forward.
2. This operation is complex to real if the flag is set to inverse.
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input/output buffer is fully aligned for the target hardware.
5. The complex buffer is in Complex Interleaved compact format.
6. This operation is supported in both single and double precision.

rfftfP

Real to Complex Forward FFT

Function Definitions

```
void rfftf(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    float *rso,
                    long   n);

void rfftf_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    float *rso,
                    long  *n);

void rfftfd(
                    double *rv1,
                    long    rv1inc,
                    double *cvo,
                    long    cvoinc,
                    double *rso,
                    long    n);

void rfftfd_(
                    double *rv1,
                    long   *rv1inc,
                    double *cvo,
                    long   *cvoinc,
                    double *rso,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
rso	output	Nyquist value
n	input	FFT size (number of real points)

Return Value

None: void function.

Description

The forward real-to-complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rfftfnP

Real to Complex Forward FFT (Non-Destructive)

Function Definitions

```
void rfftfn(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    float *rso,
                    long   n);

void rfftfn_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    float *rso,
                    long  *n);

void rfftfnd(
                    double *rv1,
                    long    rv1inc,
                    double *cvo,
                    long    cvoinc,
                    double *rso,
                    long    n);

void rfftfnd_(
                    double *rv1,
                    long   *rv1inc,
                    double *cvo,
                    long   *cvoinc,
                    double *rso,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
cvo	output	Complex output vector
cvoinc	input	Output vector increment
rso	output	Nyquist value
n	input	FFT size (number of real points)

Return Value

None: void function.

Description

The forward real-to-complex FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rffti

Complex to Real Inverse FFT

Function Definitions

```
void rffti(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void rffti_(
                    float *cv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
rs1	input	Nyquist point (real part of cv1[n/2])
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	FFT size (number of real output points)

Return Value

None: void function.

Description

The inverse complex-to-real FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.

6. This operation is supported in float only (no double precision).

rfftinP

Complex to Real Inverse FFT (Non-Destructive)

Function Definitions

```
void rfftin(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void rfftin_(
                    float *cv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void rfftind(
                    double *cv1,
                    long    cv1inc,
                    double *rs1,
                    double *rvo,
                    long    cvoinc,
                    long    n);

void rfftind_(
                    double *cv1,
                    long   *cv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
rs1	input	Nyquist point (real part of cv1[n/2])
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	FFT size (number of real output points)

Return Value

None: void function.

Description

The inverse complex-to-real FFT of the input vector is computed and stored in the output vector.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rfftscP

Real FFT Scale and Format

Function Definitions

```
void rfftsc(
                    float *cvio,
                    long   n,
                    long   flg1,
                    long   flg2);

void rfftsc_(
                    float *cvio,
                    long  *n,
                    long  *flg1,
                    long  *flg2);

void rfftscd(
                    double *cvio,
                    long    n,
                    long    flg1,
                    long    flg2);

void rfftscd_(
                    double *cvio,
                    long   *n,
                    long   *flg1,
                    long   *flg2);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cvio	in/output	Complex input/output vector
n	input	Number of real points in cvio (n/2 complex)
flg1	input	Format flag
flg2	input	Scale flag

Return Value

None: void function.

Description

Scales and formats real FFT input/output vector, converting between pure complex form ($\mathbf{n/2+1}$ complex points) and a packed RFFT form ($\mathbf{n/2}$ complex points, with real part of **cvio n/2** packed into the imaginary part of **cvio0**

Restrictions and Notes

1. The FFT is performed faster if the FFT lengths are a power of two.

2. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
3. Complex buffers are in Complex Interleaved format.
4. This operation is supported in both single and double precision.

rft2fcP

2D Real to Complex Forward FFT - Column Compact

Function Definitions

```
void rft2fc(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc);

void rft2fc_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void rft2fcd(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc);

void rft2fcd_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional real FFT is defined as a series of forward real-to-complex FFTs down the columns, followed by forward complex-to-complex FFTs across the rows. The real FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rft2frP

2D Real to Complex Forward FFT - Row Compact

Function Definitions

```
void rft2fr(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc);

void rft2fr_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc);

void rft2frd(
                    float *rv1,
                    float *cvn,
                    long   cvninc,
                    float *cvo,
                    long   nr,
                    long   nc);

void rft2frd_(
                    float *rv1,
                    float *cvn,
                    long  *cvninc,
                    float *cvo,
                    long  *nr,
                    long  *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvn	output	Complex Nyquist vector
cvninc	input	Nyquist vector increment
cvo	output	Complex output vector
nr	input	Number of rows in rv1
nc	input	Number of columns in rv1

Return Value

None: void function.

Description

The two dimensional forward real FFT of the input vector is computed and stored in the output vector. The rows are assumed to be compact. The two dimensional real FFT is defined as a series of forward real-to-complex FFTs across the rows, followed by forward complex-to-complex FFTs down the columns. The real FFTs each produce a single complex Nyquist frequency output, which is stored in cvn. A final complex-to-complex FFT of cvn is computed.

Restrictions and Notes

1. The strided elements in the real input buffer must be twice as big as the number of complex elements in the output buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rft2ic

2D Complex to Real Inverse FFT - Column Compact

Function Definitions

```
void rft2ic(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc);

void rft2ic_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc);

void rft2icd(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc);

void rft2icd_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input vector is computed and stored in the output vector. The columns are assumed to be compact. The two dimensional real FFT is defined as a series of inverse complex-to-complex FFTs across the rows and input vector cvn, followed by inverse complex-to-real FFTs down the columns. The real FFTs each require a single real Nyquist frequency input, which is read from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in float only (no double precision).

rft2irP

2D Complex to Real Inverse FFT - Row Compact

Function Definitions

```
void rft2ir(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc);

void rft2ir_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc);

void rft2icd(
                    float *cv1,
                    float *cvn,
                    long   cvninc,
                    float *rvo,
                    long   nr,
                    long   nc);

void rft2ird_(
                    float *cv1,
                    float *cvn,
                    long  *cvninc,
                    float *rvo,
                    long  *nr,
                    long  *nc);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cvn	input	Complex Nyquist vector
cvninc	input	Nyquist vector increment
rvo	output	Real output vector
nr	input	Number of rows in rvo
nc	input	Number of columns in rvo

Return Value

None: void function.

Description

The two dimensional inverse real FFT of the input vector is computed and stored in the output vector. The rows are assumed to be compact. The two dimensional real FFT is defined as a series of inverse complex-to-complex FFTs down the columns and input vector cvn, followed by inverse complex-to-real FFTs across the rows. The real FFTs each require a single real Nyquist frequency input, which is read from the inverse FFT of cvn.

Restrictions and Notes

1. The strided elements in the real output buffer must be twice as big as the number of complex elements in the input buffer.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
3. The FFT is performed faster if the FFT lengths are a power of two.
4. The FFT is performed faster if the input and output buffers are fully aligned for the target hardware.
5. Complex buffers are in Complex Interleaved format.
6. This operation is supported in both single and double precision.

rfzcP

Find Location of First Zero Crossing

Function Definitions

```
void rfzc(
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    long  *idx);

void rfzc_(
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    long  *idx);

void rfzcd(
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    long   *idx);

void rfzcd_(
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    long   *idx);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
n	input	Number of points in input vector
idx	output	Vector index of first zero crossing

Return Value

None: void function.

Description

The vector rv1 is searched for zero crossings, instances where a negative element is preceded or followed by a positive element. The position of the first crossing is returned in idx. The position is the index within the examined elements only and is dependent on the increment used. The first index is 0. If no crossings are found, -1 is returned in idx.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

rgesP

Find Location of First Element Greater/Equal to a Scalar

Function Definitions

```
void rges(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n,
                    long  *idx);

void rges_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n,
                    long  *idx);

void rgesd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    long    n,
                    long   *idx);

void rgesd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    long   *n,
                    long   *idx);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar input
n	input	Number of points in output vector
idx	output	Vector index

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for an element greater or equal to the scalar rs1. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0. If no element is found meeting the criteria, -1 is returned for idx.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rgtsP

Find Location of First Element Greater Than a Scalar

Function Definitions

```
void rgts(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n,
                    long  *idx);

void rgts_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n,
                    long  *idx);

void rgtsd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    long    n,
                    long   *idx);

void rgtsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    long   *n,
                    long   *idx);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar input
n	input	Number of points in output vector
idx	output	Vector index

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for an element greater than the scalar rs1. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0. If no element is found meeting the criteria, -1 is returned for idx.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rltsP

Find Location of First Element Less Than a Scalar

Function Definitions

```
void rlts(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n,
                    long  *idx);

void rlts_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n,
                    long  *idx);

void rltsd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    long    n,
                    long   *idx);

void rltsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    long   *n,
                    long   *idx);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar input
n	input	Number of points in output vector
idx	output	Vector index

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for an element less than the scalar rs1. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0. If no element is found meeting the criteria, -1 is returned for idx.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rmaxP

Find the Real Maximum Value and its Location

Function Definitions

```
void rmax(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void rmax_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void rmaxd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void rmaxd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Maximum value found
idx	output	Location of maximum
n	input	Number of points to scan

Return Value

None: void function.

Description

The vector rv1 is searched and the maximum value found is returned in rso. The position of this element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rmaxmgP

Find the Real Maximum Magnitude

Function Definitions

```
void rmaxmg(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void rmaxmg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void rmaxmgd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void rmaxmgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rso	output	Maximum magnitude found
idx	output	Index of element with max magnitude
n	input	Number of points in output vector

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the greatest magnitude. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rminP

Find the Real Minimum Value and its Position

Function Definitions

```
void rmin(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void rmin_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void rmind(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void rmind_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Minimum value found
idx	output	Position of minimum
n	input	Number of points to scan

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the least value. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rminmgP

Find the Real Minimum Magnitude

Function Definitions

```
void rminmg(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void rminmg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void rminmgd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void rminmgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rso	output	Minimum magnitude found
idx	output	Position of element found
n	input	Number of points in output vector

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for the element with the least magnitude. The position of this element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rmsqvP

Root Mean Square of a Real Vector

Function Definitions

```
void rmsqv(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n);

void rmsqv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n);

void rmsqvd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    long    n);

void rmsqvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points in output vector

Return Value

None: void function.

Description

The square root of the mean value of the squares of the processed elements of rv1 is returned in rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rnesP

Find Position of First Element Not Equal to a Scalar

Function Definitions

```
void rnes(
                    float *rv1,
                    long   rv1inc,
                    float *rs0,
                    long  *idx,
                    long   n);

void rnes_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs0,
                    long  *idx,
                    long  *n);

void rnesd(
                    double *rv1,
                    long    rv1inc,
                    double *rs0,
                    long   *idx,
                    long    n);

void rnesd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs0,
                    long   *idx,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar input
n	input	Number of points to examine
idx	output	Vector index

Return Value

None: void function.

Description

The vector rv1 is searched from the beginning for an element not equal to the scalar rs1. The position of the found element is returned in idx, but is not a traditional offset. It is the position within the examined elements only and is dependent on the increment used. The first position is 0. If no element is found meeting the criteria, -1 is returned for idx

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

rsveP

Running Sum of Real Vector

Function Definitions

```
void rsve(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   no,
                    long   nw);

void rsve_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *no,
                    long  *nw);

void rsved(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    no,
                    long    nw);

void rsved_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *no,
                    long   *nw);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	output	Real scalar input
rvo	input	Real output vector
rvoinc	input	Output vector increment
no	input	Number of sums in output vector
nw	input	Number of points to sum in running window

Return Value

None: void function.

Description

The running sum of the elements of rv1, scaled by rs1, is written to rvo. Each output element is the sum of nw elements centered around the corresponding input point. The first nw/2 output points are computed over a variable window with a starting width of nw/2 elements, increasing by one for each output point, to a width of nw. The last nw/2 output points are similarly computed over a window tapering down from nw to nw/2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

shphuP

Schafer's Phase Unwrapping

Function Definitions

```
void shphu(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void shphu_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void shphud(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void shphud_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	input	Real comparative scalar
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Input vector rv1 is assumed to be a vector of phase angles, in units of radians. The output vector contains the phase angles with multiples of 2*PI added or subtracted, based on the difference between adjoining points relative to an input threshold, rs1. This has the effect of "unwrapping" the discontinuities caused by phases having been limited to magnitudes of 2*PI during a previous computation.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.

3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

shphufP

Schafer's Phase Unwrapping, Fraction of a Circle Argument

Function Definitions

```
void shphuf(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void shphuf_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void shphufd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void shphufd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rs1	input	Real comparative scalar
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Input vector rv1 is assumed to be a vector of phase angles, in units of fractions of a circle. The output vector contains the phase angles with multiples of 1.0 added or subtracted, based on the difference between adjoining points relative to an input threshold, rs1. This has the effect of "unwrapping" the discontinuities caused by phases having being limited to magnitudes of 1.0 by a previous computation.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.

3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

sn2P

Sum the Squared Difference Between Two Vectors

Function Definitions

```
void sn2(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rso,
                    long   n);

void sn2_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rso,
                    float *n);

void sn2d(
                    double *cv1,
                    long    cv1inc,
                    double *cv2,
                    long    cv2inv,
                    double *rso,
                    long    n);

void sn2d_(
                    double *cv1,
                    long   *cv1inc,
                    double *cv2,
                    long   *cv2inv,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Second real input vector 2
rv2inc	input	Vector increment for rv2
rso	output	Real scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

The sum of the squares of the differences between vectors rv1 and rv2 is written to rso.

Restrictions and Notes

1. The strided elements in the two input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

stddevc

Standard Deviation of Complex Vector

Function Definitions

```
void stddevc(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    long   n);

void stddevc_(
                    float *cv1,
                    long  *cv1inc,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
rs1	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the standard deviation of the input complex vector. The result is always real. The standard deviation is the square root of the variance.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).
5. Complex buffers are in Complex Interleaved format.

stddevv

Standard Deviation of Real Vector

Function Definitions

```
void stddevv(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n);

void stddevv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points to process

Return Value

None: void function.

Description

Computes standard deviation using the two-pass approach, which is more stable for long vectors.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

svdivP

Divide Scalar by Vector

Function Definitions

```
void svdiv(
                    float *rs1,
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void svdiv_(
                    float *rs1,
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void svdivd(
                    double *rs1,
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void svdivd_(
                    double *rs1,
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rs1	input	Real scalar dividend
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Divides the real scalar by the selected points in the input vector and stores the result in the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

sveP

Sum of Real Vector

Function Definitions

```
void sve(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void sve_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void sved(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void sved_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of points to sum

Return Value

None: void function.

Description

The sum of the elements of rv1 is written to rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

svemgP

Sum of Vector Magnitudes

Function Definitions

```
void svemg(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void svemg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void svemgd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void svemgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

Sum of absolute values of rv1 is returned in rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

svesqP

Sum of Vector Elements Squared

Function Definitions

```
void svesq(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void svesq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void svemsq(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void svemsq_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

Sum of the squares of elements of rv1 is returned in rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

svessqP

Sum of Vector Element Signed Squares

Function Definitions

```
void svessq(
                    float *rv1,
                    long   rv1inc,
                    float *rso,
                    long   n);

void svessq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rso,
                    long  *n);

void svessqd(
                    double *rv1,
                    long    rv1inc,
                    double *rso,
                    long    n);

void svessqd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rso,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rso	output	Real scalar output
n	input	Number of samples to process

Return Value

None: void function.

Description

The sum of the signed squares of the elements of rv1 is written to rso.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

tconvP

Tapered Convolution

Function Definitions

```
void tconv(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n0,
                    long   n2,
                    long   n1);

void tconv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n0,
                    long  *n2,
                    long  *n1);

void tconvd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n0,
                    long    n2,
                    long    n1);

void tconvd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n0,
                    long   *n2,
                    long   *n1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Second real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Output vector increment
no	input	Number of points to output
n2	input	Number of points in rv2 (kernel)
n1	input	Number of points in rv1

Return Value

None: void function.

Description

tconv convolves the vector rv1 with the vector rv2 or correlates vectors rv1 with rv2 depending on the sign of inc1 and inc2. Vector rv1 is treated as padded with zeros as necessary to allow the tapering effect to occur.

Restrictions and Notes

1. The strided elements in the input vector **rv1** must be equal to or greater than **n1** in length.
2. The strided elements in the input vector **rv2** must be equal to or greater than **n2** in length.
3. The strided elements in the output vector **rvo** must be equal to or greater than **no** in length.
4. The operation is performed faster when **n1**, **n2** and **no** are set to a power of two.
5. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
6. This operation is supported in both single and double precision.

Thread_SetParams

Set Number of Threads.

Function Definitions

```
void Thread_SetParams(
                      int num_threads,
                      int max_num_running);
```

Parameters

Argument	I/O	Description
num_threads	input	Maximum number of threads.
max_num_running	input	Maximum number of threads left running.

Return Value

None: void function.

Description

The Threaded Vector library is multithreaded and will take advantage of multiple cores on the processor invoking it. Utilising multiple threads is automatic:

- The maximum number of threads used is set when the Vector initialization function is called.
- The maximum number running at any one time is also set at that point. If a threaded routine is called with (say) 4 threads and we have hit this maximum number running then four of them are shut down before the new function is executed.
- The number of threads invoked when a routine is called, is decided by that routine by reference to the data (vector or matrix) size specified in the call, to provide the best performance for that call.

Restrictions and Notes

1. A threaded, and a non-threaded ("serial") version of the library are provided. If you wish to only ever use one thread in a library call, use the serial version of the library.
2. The maximum number of threads used for a specific function call, and the maximum number kept running at any one time, can be changed by a call to the routine Thread_SetParams with arguments num_threads and max_num_running. This call, if used, *must* be made before the Vector initialization function is called.
3. When calling the routine Thread_SetParams the value of max_num_running must be greater or equal to 3*num_threads. If the user enters a smaller value than this in their Thread_SetParams function call then the function will set the value of max_num_running to 3*num_threads.
4. If no call to Thread_SetParams is made, the library default values will be utilised.
5. This function is now defined in the Vector library include files. Therefore, it may be called without defining it as an external function in the application code.

transP

Complex Vector Divided by Real Vector (Transfer)

Function Definitions

```
void trans(
                    float *rv1,
                    float *cv1,
                    float *cvo,
                    long   n);

void trans_(
                    float *rv1,
                    float *cv1,
                    float *cvo,
                    long  *n);

void transd(
                    double *rv1,
                    double *cv1,
                    double *cvo,
                    long    n);

void transd_(
                    double *rv1,
                    double *cv1,
                    double *cvo,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cv1	input	Complex input vector
cvo	output	Complex output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element of the complex input vector, cv1, is divided by the corresponding element in the real vector, rv1, and copied to the complex output vector, cvo. (This routine is included only for backwards compatibility with other subroutine libraries. For new programs, use crvdiv.)

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

vaamP

Vector Add, Add, and Multiply

Function Definitions

```
void vaam(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vaam_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vaamd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vaamd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Vector increment for rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment for rv2
`rv3`	input	Real input vector 3
`rv3inc`	input	Vector increment for rv3
`rv4`	input	Real input vector 4
`rv4inc`	input	Vector increment for rv4
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of output samples to process

Return Value

None: void function.

Description

Adds real input vectors rv1 and rv2, then multiplies by the sum of input vectors rv3 and rv4, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vabsP

Vector Absolute Value

Function Definitions

```
void vabs(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vabs_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vabsd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vabsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the absolute value of each element in the real input vector and places the results in the real output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vabserr

Absolute Error

Function Definitions

```
void vabserr(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long   n);

void vabserr_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
avg	output	Average error
max	output	Maximum error
i	output	Index where max occurred
n	input	Number of samples to process

Return Value

None: void function.

Description

The absolute error computes fabs $(rv1k - rv2k)$ for all k elements in each vector. The average of this difference is computed and stored at *avg. The maximum difference is stored at *max, and the index to the maximum value is stored at *i.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vacosP

Vector Arccosine.

Function Definitions

```
void vacos(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vacos_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vacosd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vacosd_(
                    double *rv1,
                    long    eA,
                    double *R,
                    long    strideR,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compute the arccosine for each element of a vector. Element angle values are in radians.
$\mathtt{R}[j * \mathtt{strideR}] :=$ acos$(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vaddP

Add Two Vectors.

Function Definitions

```
void vadd(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vadd_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vaddd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vaddd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Compute the arccosine for each element of a vector. Element angle values are in radians.
$\mathtt{R}[j * \mathtt{strideR}] :=$ acos$(\mathtt{A}[j * \mathtt{strideA}])$ where $0 \leqslant j < n$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

1. Input and output buffers must all be the same size.
2. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).

Notes

Input arguments are expressed in radians.

vaddsub

Real Add/Subtract

Function Definitions

```
void vaddsub(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvs,
                    long   rvsinc,
                    float *rvd,
                    long   rvdinc,
                    long   n);

void vaddsub_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvs,
                    long  *rvsinc,
                    float *rvd,
                    long  *rvdinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
rvs	output	real sum
incs	input	stride for rvs
rvd	output	real difference
incd	input	stride for rvd
n	input	Number of samples to process

Return Value

None: void function.

Description

This function adds elements from rv1 and rv2 placing the result in rvs. Concurrently it computes rv1 - rv2, placing the difference in rvd. This function may be used in-place.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vaintP

Vector Align to Integer

Function Definitions

```
void vaint(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vaint_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vaintd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vaintd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Truncates each element of the real input vector rv1 and places the integer portion in the real output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vamP

Vector Add and Multiply

Function Definitions

```
void vam(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vam_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vamd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vamd_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Adds real input vectors rv1 and rv2, then multiplies by input vector rv3, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vanintP

Vector Align to Nearest Integer

Function Definitions

```
void vanint(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vanint_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vanintd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vanintd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Rounds each element of the real input vector rv1 and places the integer portion in the real output vector rvo. If the fractional portion is equal to.5, the output is rounded to the nearest even integer

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

variancec

Variance of Complex Vector

Function Definitions

```
void variancec(
                    float *cv1,
                    long   cv1inc,
                    float *rs1,
                    long   n);

void variancec_(
                    float *cv1,
                    long  *cv1inc,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
cv1	input	Input complex vector
cv1inc	input	Input vector increment
rs1	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the variance of the input complex vector. The result is always real. This function first computes the mean of the vector. Then, for each point in the vector, the square of the difference between the real and imaginary components of rv1k and the mean is accumulated. The result is the average of this value.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

variancev

Variance of Real Vector

Function Definitions

```
void variancev(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    long   n);

void variancev_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rs1	output	Where to deposit real result
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the variance of the input vector. This function first computes the mean of the vector. Then, for each point in the vector, the square of the difference between rv1k and the mean is accumulated. The result is the average of this value.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vasbmP

Vector Add, Subtract, and Multiply

Function Definitions

```
void vasbm(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vasbm_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vasbmd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vasbmd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Vector increment for rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment for rv2
`rv3`	input	Real input vector 3
`rv3inc`	input	Vector increment for rv3
`rv4`	input	Real input vector 4
`rv4inc`	input	Vector increment for rv4
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Adds real input vectors rv1 and rv2, then multiplies by the difference of input vectors rv3 and rv4, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vasinP

Vector Arcsine.

Function Definitions

```
void vasin(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vasin_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vasind(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vasind_(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compute the arcsine for each element of a vector. Element angle values are in radians.
$\texttt{rvo}[j * \texttt{rvoinc}] :=$ asin$(\texttt{rv1}[j * \texttt{rv1inc}])$ where $0 \leqslant j < n$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vasmP

Vector Add and Scalar Multiply

Function Definitions

```
void vasm(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vasm_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vasmd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vasmd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Adds the input real vector rv1 to real vector rv2, then multiplies by real scalar rs1, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vatanP

Vector Arctangent.

Function Definitions

```
void vatan(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vatan_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vatand(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vatand_(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compute the arctangent for each element of a vector. Element angle values are in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vatan2P

Vector Arctangent of Two Arguments

Function Definitions

```
void vatan2(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vatan2_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vatan2d(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vatan2d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the arctangent of the fraction created by dividing each element of the first real input vector by the corresponding element of the second real input vector. The result is placed in the real output vector. Results are in radians, from $-Pi$ to $+Pi$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vatn2fP

Vector Arctangent of Two Arguments in Fractions

Function Definitions

```
void vatn2f(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vatn2f_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vatn2fd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vatn2fd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the arctangent of the fraction created by dividing each element of the first real input vector by the corresponding element of the second real input vector. The result is placed in the real output vector. Results are in radians, from -0.5 to +0.5.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vavexpP

Vector Exponential Averaging

Function Definitions

```
void vavexp(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vavexp_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vavexpd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vavexpd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rs1	input	Real input scalar
rvo	in/output	Real output vector
rvoinc	output	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Calculates the exponential average of vector rvo and vector rv1 using rs1 as a time constant.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vavg

Average

Function Definitions

```
void vavg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vavg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the average of corresponding elements of rv1 and rv2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vavgnegy

Average x, -y

Function Definitions

```
void vavgnegy(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vavgnegy_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the average of corresponding elements of rv1 and -rv2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vavlinP

Vector Linear Averaging

Function Definitions

```
void vavlin(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vavlin_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vavlind(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vavlind_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rs1	input	Real input scalar
rvo	in/output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of output points

Return Value

None: void function.

Description

Calculates the linear average of vector rvo and vector rv1 using rs1 as the number of vectors averaged over. The result is placed in rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vaxyplusz

Real aXY + Z

Function Definitions

```
void vaxyplusz(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vaxyplusz_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rvx	input	Input real vector
incx	input	Input vector increment
rvy	input	Input real vector
incy	input	Input vector increment
rvz	input	Input real vector
incz	input	Input vector increment
a	input	Input real scalar
vout	output	Output real vector
inco	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes aXY + Z for a real scalar a and real vectors X, Y and Z.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vceil

Ceiling

Function Definitions

```
void vceil(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vceil_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the ceil for each input vector element. The ceil of x is x if x is an integer, otherwise it is the truncated value of x+1. For example, ceil(2) = 2; ceil (-3.1) = - 3, ceil (4.1) = 5.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vclipP

Vector Clip

Function Definitions

```
void vclip(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vclip_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar minimum value
rs2	input	Real scalar maximum value
rvo	output	Real vector output
rvoinc	input	Vector increment
n	input	Number of points to process

Return Value

None: void function.

Description

Compare vector points against minimum value rs1 and maximum value rs2, if less than rs1, use rs1, if greater than rs2, use rs2; else set rvo to input vector point value. Do this for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vcliph

Clip High

Function Definitions

```
void vcliph(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcliph_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
h	input	Pointer to high limit
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function clips the input vector: values >= *h are replaced with *h.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vclipl

Clip Low

Function Definitions

```
void vclipl(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vclipl_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
lo	input	Pointer to low limit
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function clips the input vector: values $<$ *lo are replaced with *lo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vclrP

Zero a Vector

Function Definitions

```
void vclr(
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vclr_(
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rvo	output	Real output vector
rvoinc	input	Real output vector sample increment
n	input	Number of points to clear

Return Value

None: void function.

Description

Zeros are written to the vector rvo.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vcmprsP

Vector Compress

Function Definitions

```
void vcmprs(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmprs_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcmprsd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vcmprsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector Increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector Increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Each point in input vector rv2 is compared to zero. If the value in rv2 is non-zero, the corresponding point from input vector rv1 is put into corresponding point in output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vcmpXX

Compare Vectors

Function Definitions

```
void vcmpeq(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmpeq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcmpne(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmpne_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcmpge(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmpge_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

```
void vcmpgt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmpgt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcmple(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmple_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcmplt(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcmplt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Input real vector
`rv1inc`	input	Input vector increment
`rv2`	input	Input complex vector
`rv3inc`	input	Input real increment
`rvo`	output	Output vector
`rvoinc`	input	Output vector increment
`n`	input	Number of samples to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2, leaving a 32 bit long flag value in the corresponding elements of rv3. If the condition is true, the resulting flag value is 0xffff_ffff. If the resulting condition is false, the resulting flag value is 0x0000_0000. The resulting flag vectors may then be operated upon by other long integer vector functions if desired.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vcosP

Vector Cosine

Function Definitions

```
void vcos(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcos_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcosd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vcosd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the cosine of each element in the real input vector and places the result in the real output vector. The elements of the input vector are assumed to be in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vcosfP

Vector Cosine in Fractions

Function Definitions

```
void vcosf(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vcosf_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vcosfd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vcosfd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the cosine of each element of the real input vector and places the results in the real output vector. The elements of the input vector rv1 are assumed to be in fractions of a circle.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vdbpwrP

Vector Conversion to dB

Function Definitions

```
void vdbpwr(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vdbpwr_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vdbpwrd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vdbpwrd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rs1	input	Real scalar (0 dB value)
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the power in decibels of input real vector rv1, relative to real scalar rs1, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vdivP

Divide One Vector by Another

Function Definitions

```
void vdiv(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vdiv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vdivd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vdivd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the second input vector are divided by points from the first vector and the results are placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vdpsp

Vector Convert Double to Single Precision

Function Definitions

```
void vdpsp(
                    float *dv1,
                    long   dv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vdpsp_(
                    float *dv1,
                    long  *dv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
dv1	input	Double precision real input vector
dv1inc	input	Increment for dv1
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of points to move

Return Value

None: void function.

Description

Converts the input double precision floating point vector dv1 to single precision floating point and puts the result in output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only.

veqv

Exclusive NOR (XNOR) Two Integer Vectors

Function Definitions

```
void veqv(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void veqv_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are exclusively NORed (XNORed) and placed into the output vector. Each bit in the output vector will be set if both inputs have that bit set, or both have it clear.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.

veucl2P

Vector Euclidean Distance

Function Definitions

```
void veucl2(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void veucl2_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void veucl2d(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void veucl2d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector Increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector Increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the square root of the sum of the squares of the two input vectors, rv1 and rv2 and stores the results in output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

veucl3P

Vector Euclidean Distance (3 Dimensional)

Function Definitions

```
void veucl3(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void veucl3_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void veucl3d(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void veucl3d_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the three dimensional distance of each point, using the three input vectors and writing the output to vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vexpP

Vector Exponentiation

Function Definitions

```
void vexp(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vexp_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vexp(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vexp_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv	input	Real input vector
inc	input	Vector increment for rv
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Raises the mathematical constant 'e' to the power of the elements in rv, for n points. The results are written to output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vexp2P

Vector Base 2 Exponential

Function Definitions

```
void vexp2(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vexp2_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vexp2d(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vexp2d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the base 2 exponential of input real vector rv1 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vexp10P

Vector Base 10 Exponential

Function Definitions

```
void vexp10(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vexp10_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vexp10d(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vexp10d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the base 10 exponential of input real vector rv1 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vexp2est

exp2 Estimate

Function Definitions

```
void vexp2est(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vexp2est_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function estimates 2x for each input value x. Estimate accuracy generally has a relative error < 1/16th. This function takes advantage of high-performance processor instructions that compute the estimate.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only.

vfillP

Set a Vector in Memory to a Scalar Value

Function Definitions

```
void vfill(
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vfill_(
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vfilld(
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vfilld_(
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rs1	input	Real scalar
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of points to fill

Return Value

None: void function.

Description

The scalar value rs1 is copied to the elements of output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vfindlastXX

Find Element in a Vector

Function Definitions

```
void vfindlasteq(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindlasteq_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindlastne(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindlastne_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindlastge(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindlastge_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindlastgt(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindlastgt_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);
```

```
void vfindlastle(
                        float *rv1,
                        long   rv1inc,
                        float *k,
                        long  *i,
                        long   n);

void vfindlastle_(
                        float *rv1,
                        long  *rv1inc,
                        float *k,
                        long  *i,
                        long  *n);

void vfindlastlt(
                        float *rv1,
                        long   rv1inc,
                        float *k,
                        long  *i,
                        long   n);

void vfindlastlt_(
                        float *rv1,
                        long  *rv1inc,
                        float *k,
                        long  *i,
                        long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
k	input	value for which to search
i	output	Resulting index value
n	input	Number of samples to process

Return Value

None: void function.

Description

These functions are identical to vfindxx, except that the vector is searched from right to left. See âĂIJvfindxx Find Element in a VectorâĂİ.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vfindXX

Find Element in a Vector

Function Definitions

```
void vfindeq(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindeq_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindne(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindne_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindge(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindge_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);

void vfindgt(
                    float *rv1,
                    long   rv1inc,
                    float *k,
                    long  *i,
                    long   n);

void vfindgt_(
                    float *rv1,
                    long  *rv1inc,
                    float *k,
                    long  *i,
                    long  *n);
```

```
void vfindle(
                        float *rv1,
                        long   rv1inc,
                        float *k,
                        long  *i,
                        long   n);

void vfindle_(
                        float *rv1,
                        long  *rv1inc,
                        float *k,
                        long  *i,
                        long  *n);

void vfindlt(
                        float *rv1,
                        long   rv1inc,
                        float *k,
                        long  *i,
                        long   n);

void vfindlt_(
                        float *rv1,
                        long  *rv1inc,
                        float *k,
                        long  *i,
                        long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
k	input	value for which to search
i	output	Resulting index value
n	input	Number of samples to process

Return Value

None: void function.

Description

vfindeq: This function searches the real vector rv1 from left to right until a value equal to k is located. If no value matches, this function sets *i to -1. Otherwise this sets *i to the index of the value meeting the search criteria. If multiple items match, the index of the first matching value is returned.

- vfindne searches for the first value not equal to k.
- vfindge searches for the first value greater than or equal to k.
- vfindgt searches for the first value greater than k.
- vfindle searches for the first value less than or equal to k.
- vfindlt searches for the first value less than k.

Restrictions and Notes

1. The strided elements in the input vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vector is fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vfloor

Floor

Function Definitions

```
void vfloor(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vfloor_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the floor for each input vector element. The floor of x is x if x is an integer, otherwise it is the truncated value of x-1. For example, floor(2) = 2; floor (-3.1) = -4, floor (4.1) = 4.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in single precision only (no double precision).

vfracP

Vector Truncate to Fraction

Function Definitions

```
void vfrac(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vfrac_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vfracd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vfracd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Truncates each element of the real input vector and places the fractional portion in the real output vector. Result values will be between -1.0 and 1.0.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vfracnP

Vector Truncate to Nearest Fraction

Function Definitions

```
void vfracn(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vfracn_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vfracnd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vfracnd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Rounds each element of the real input vector and places the fractional portion in the real output vector. If the fractional portion is equal to .5, the output will reflect rounding to the nearest even integer. The real output vector is in the range -0.5 to +0.5.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vgathrP

Vector Gather

Function Definitions

```
void vgathr(
                    float *rv1,
                    float *iv2,
                    long   iv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vgathr_(
                    float *rv1,
                    float *iv2,
                    long  *iv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vgathrd(
                    double *rv1,
                    double *iv2,
                    long    iv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vgathrd_(
                    double *rv1,
                    double *iv2,
                    long   *iv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input vector
iv2	input	Vector containing indices to rv1
iv2inc	input	Increment for iv2
rvo	output	Output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the input vector iv2 are used as indices into input vector rv1 to select points to be placed into output vector rvo. The first point of vector rv1 has an index of 0.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vgenP

Generate a Vector in Memory

Function Definitions

```
void vgen(
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vgen_(
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vgend(
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vgend_(
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rs1	input	Starting value
rs2	input	Ending value
rvo	input	Real output vector
rvoinc	input	Output vector sample increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Generates a linear ramp from the value of rs1 to the value of rs2 and writes the results into rvo.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

viadd

Add Two Integer Vectors

Function Definitions

```
void viadd(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void viadd_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are added and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

viand

And Two Integer Vectors

Function Definitions

```
void viand(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void viand_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are anded and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

viars

Vector Integer Arithmetic Right Shift

Function Definitions

```
void viars(
                    long *iv1,
                    long  iv1inc,
                    long *is1,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void viars_(
                    long *iv1,
                    long *iv1inc,
                    long *is1,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
is1	input	Integer scalar shift count
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first input vector are arithmetically shifted right by the shift count, is1, and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

viclipP

Vector Inverse Clip

Function Definitions

```
void viclip(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void viclip_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void viclipd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void viclipd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar minimum value
rs2	input	Real scalar maximum value
rvo	output	Real vector output
rvoinc	input	Vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Compare vector points; if negative, takes the smaller of input vector point rv1 and value rs1, if non-negative, takes the larger of input vector point rv1 and value rs2. The results are stored to output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vils

Vector Integer Left Shift

Function Definitions

```
void vils(
                    long *iv1,
                    long  iv1inc,
                    long *is1,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vils_(
                    long *iv1,
                    long *iv1inc,
                    long *is1,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
is1	input	Integer scalar shift count
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first input vector are shifted left by the shift count, is1, and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

vimagP

Extract Imaginary Part of Complex Vector

Function Definitions

```
void vimag(
                    float *cv1,
                    long   cv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vimag_(
                    float *cv1,
                    long  *cv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vimagd(
                    double *cv1,
                    long    cv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vimagd_(
                    double *cv1,
                    long   *cv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Input vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points in output vector

Return Value

None: void function.

Description

Extract the imaginary part of complex vector cv1 and write results to output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

vifill

Integer Vector Fill

Function Definitions

```
void vifill(
                    long *is1,
                    long *iv0,
                    long  rvoinc,
                    long  n);

void vifill_(
                    long *is1,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
rs1	input	pointer to input fill value
rvo	output	output long integer vector
rvoinc	input	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function fills the output vector with the integer value at *rs1.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This is an integer operation.

vimov

Integer Vector Move

Function Definitions

```
void vimov(
                    long *iv1,
                    long  iv1inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vimov_(
                    long *iv1,
                    long *iv1inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
rv1	input	starting address of long integer vector 1
rv1inc	input	increment for vector 1
rvo	output	output long integer vector
rvoinc	input	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function copies long integer elements from one vector to another.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

vimul

Multiply Two Integer Vectors

Function Definitions

```
void vimul(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vimul_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are multiplied and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

vindexP

Vector Index, Truncate

Function Definitions

```
void vindex(
                    float *rv1,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vindex_(
                    float *rv1,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vindexd(
                    double *rv1,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vindexd_(
                    double *rv1,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input vector
rv2	input	Vector containing indices
rv2inc	input	Increment for rv2
rvo	output	Output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the input vector rv2 are converted to integer by truncation. These integers are then used as indices into input vector rv1 for points to be placed into the output vector rvo. The first element of input vector rv1 is considered to be at index 1, not 0.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.

4. This operation is supported in both single and double precision.
5. Complex buffers are in Complex Interleaved format.

vineg

Vector Integer Negate

Function Definitions

```
void vineg(
                    long *iv1,
                    long  iv1inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vineg_(
                    long *iv1,
                    long *iv1inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first input vector are negated and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

vintbP

Vector Interpolate

Function Definitions

```
void vintb(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vintb_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vintbd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vintbd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rs1	input	Real scalar interpolation factor
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of output samples to process

Return Value

None: void function.

Description

Does a linear interpolation between real input vectors rv1 and rv2 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vintegrate

Discrete Integrator

Function Definitions

```
void vintegrate(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vintegrate_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

For each element rvok in the output vector, this function computes the sum of the first k entries in the input vector rv1. This function is sometimes called a "running sum".

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

vior

Vector Integer OR Two Integer Vectors

Function Definitions

```
void vior(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vior_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are ORed and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

viramp

Integer Ramp Generate

Function Definitions

```
void viramp(
                    long *is1,
                    long *is2,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void viramp_(
                    long *is1,
                    long *is2,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
is1	input	pointer to starting value
is2	input	pointer to step size
iv	output	output long integer vector
inc	input	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function generates a ramp, starting with *rs1, and incrementing by *rs2 for each element thereafter.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This is an integer operation.

virs

Vector Integer Right Shift

Function Definitions

```
void virs(
                    long *iv1,
                    long  iv1inc,
                    long *is1,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void virs_(
                    long *iv1,
                    long *iv1inc,
                    long *is1,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
is1	input	Integer scalar shift count
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first input vector are shifted right by the shift count, is1, and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

visdiv

Integer Vector Scalar Divide

Function Definitions

```
void visdiv(
                    long *iv1,
                    long  iv1inc,
                    long *is1,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void visdiv_(
                    long *iv1,
                    long *iv1inc,
                    long *is1,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
rv	input	starting address of long integer vector 1
inc	input	increment for vector 1
rs	input	scalar
cv2inc	input	output long integer vector
iso	output	output increment
n	input	number of elements to process

Return Value

None: void function.

Description

This function divides each entry in the input vector by *rs. The division is performed using signed long integer division.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

visequal

Test Vectors for Equality

Function Definitions

```
void visequal(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    long  *is0,
                    long   n);

void visequal_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    long  *is0,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
iso	output	Output long integer scalar
n	input	Number of samples to process

Return Value

None: void function.

Description

This function compares corresponding elements of rv1 and rv2. If all corresponding elements are equal, this function sets *iso to 0xffff_ffff. Otherwise this function sets *iso to 0x0. This function bails out early when it finds the first non-matching entries

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

visub

Subtract Two Integer Vectors

Function Definitions

```
void visub(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void visub_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are subtracted and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

viszero

Determine if Vector is Zero

Function Definitions

```
void viszero(
                    float *rv1,
                    long   rv1inc,
                    long  *is0,
                    long   n);

void viszero_(
                    float *rv1,
                    long  *rv1inc,
                    long  *is0,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
iso	output	Output long integer scalar
n	input	Number of samples to process

Return Value

None: void function.

Description

This function tests all elements in the input real vector to see if they are equal to zero. If all elements are zero, the resulting integer value is set to 0xffff_ffff. Otherwise, the resulting integer value is set to 0. This function bails out when it finds the first nonzero entry.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This is an integer operation.

vixor

Exclusive OR (XOR) Two Integer Vectors

Function Definitions

```
void vixor(
                    long *iv1,
                    long  iv1inc,
                    long *iv2,
                    long  iv2inc,
                    long *iv0,
                    long  iv0inc,
                    long  n);

void vixor_(
                    long *iv1,
                    long *iv1inc,
                    long *iv2,
                    long *iv2inc,
                    long *iv0,
                    long *iv0inc,
                    long *n);
```

Parameters

Argument	I/O	Description
iv1	input	First integer input vector
iv1inc	input	Increment for first input vector
iv2	input	Second integer input vector
iv2inc	input	Increment for second input vector
ivo	output	Integer output vector
ivoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are exclusively ORed (XORed) and placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This is an integer operation.

vlimP

Vector Limit

Function Definitions

```
void vlim(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlim_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vlimd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlimd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rs1	input	Real scalar threshold
rs2	input	Real magnitude of output
rvo	output	Real vector output
rvoinc	output	Vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Elements in output vector rvo are set to +rs2 or -rs2, depending on whether the corresponding elements in input vector rv1 are greater than or equal to input scalar rs1, or less than rs1, respectively.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlintP

Vector Linear Interpolate

Function Definitions

```
void vlint(
                    float *rv1,
                    long   n1,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlint_(
                    float *rv1,
                    long  *n1,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vlintd(
                    double *rv1,
                    long    n1,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlintd_(
                    double *rv1,
                    long   *n1,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real compact input vector
n1	input	Integer size of vector rv1
rv2	input	Second real input vector
rv2inc	input	Increment for second input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

A linear interpolation is performed between selected points on input vector rv1 by using selected points of input vector rv2 as an index (whole number part) and interpolation factor (fractional part). Results are put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlmergP

Vector Logical Merge

Function Definitions

```
void vlmerg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlmerg_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlmergd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlmergd_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment vector for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element in output vector rvo is filled with either an element from input vector rv1, or from input vector rv2, based on the value of the elements in input vector rv3. The input vector selected will be rv1 if the corresponding element in rv3 is non-zero, or rv2 if the element in rv3 is zero.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlogP

Compute the Natural Logarithm of a Vector

Function Definitions

```
void vlog(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlog_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vlogd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlogd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The natural logarithm of the points in vector rv1 are written to the output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlog10P

Vector Base 10 Logarithm

Function Definitions

```
void vlog10(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlog10_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vlog10d(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlog10d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the base 10 logarithm of input real vector rv1 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlog2P

Vector Base 2 Logarithm

Function Definitions

```
void vlog2(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlog2_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vlog2d(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vlog2d_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the base 2 logarithm of input real vector rv1 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vlog2est

Log2 Estimate

Function Definitions

```
void vlog2est(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vlog2est_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function estimates the log2 of each input value. Estimate accuracy generally has a relative error $< 1/16th$. This function takes advantage of high-performance processor instructions that compute the estimate.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

vmaP

Vector Multiply and Add

Function Definitions

```
void vma(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vma_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmad(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmad_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, adds input vector rv3, and puts the result in output vector rvo, for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmaxP

Vector Maximum

Function Definitions

```
void vmax(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmax_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmaxd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmaxd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Each element of output vector rvo is the larger of the corresponding element in input vector rv1 or rv2.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmaxmgP

Vector Maximum Magnitude

Function Definitions

```
void vmaxmg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmaxmg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmaxmgd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmaxmgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare absolute value of two input vector points, take larger value and put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vminP

Vector Minimum

Function Definitions

```
void vmin(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmin_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmind(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmind_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare two input vector points, take smaller value and put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vminmgP

Vector Minimum Magnitude

Function Definitions

```
void vminmg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vminmg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vminmgd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vminmgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment 1
rv2	input	Real input vector 2
rv2inc	input	Vector increment 2
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare absolute value of two input vector points, take smaller value and put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmmaP

Vector Multiply, Multiply, and Add

Function Definitions

```
void vmma(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmma_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmmad(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmmad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector
`rv1inc`	input	Vector increment vector for rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment vector for rv2
`rv3`	input	Real input vector 3
`rv3inc`	input	Vector increment vector for rv3
`rv4`	input	Real input vector 4
`rv4inc`	input	Vector increment vector for rv4
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, and adds it to the product of rv3 and rv4, and puts the result into output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmmsbP

Vector Multiply, Multiply, and Subtract

Function Definitions

```
void vmmsb(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmmsb_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmmsbd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmmsbd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Vector increment for rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment for rv2
`rv3`	input	Real input vector 3
`rv3inc`	input	Vector increment for rv3
`rv4`	input	Real input vector 4
`rv4inc`	input	Vector increment for rv4
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, and subtracts the product of rv3 and rv4. The results are placed into real output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmovP

Copy One Vector to Another

Function Definitions

```
void vmov(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmov_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmovd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmovd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of points to copy

Return Value

None: void function.

Description

The elements in vector rv1 are copied to vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmsaP

Vector Multiply and Scalar Add

Function Definitions

```
void vmsa(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmsa_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmsad(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmsad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real vector one
rv1inc	input	Increment for vector one
rv2	input	Real vector two
rv2inc	input	Increment for vector two
rs1	input	Real scalar additive factor
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of points to move

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, adds real scalar rs1, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmsbP

Vector Multiply and Subtract

Function Definitions

```
void vmsb(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmsb_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmsbd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmsbd_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, and subtracts input vector rv3, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vmulP

Multiply Two Vectors

Function Definitions

```
void vmul(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vmul_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vmuld(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vmuld_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for first input vector
rv2	input	Real input vector 2
rv2inc	input	Increment for second input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

The selected points from the first and second input vectors are multiplied and the products are placed into the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vnabsP

Vector Negative Absolute Value

Function Definitions

```
void vnabs(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vnabs_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vnabsd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vnabsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Takes the negation of the absolute value of each element in the real input vector and places it in the real output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vnegP

Negate a Vector

Function Definitions

```
void vneg(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vneg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vnegd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vnegd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real vector to negate
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Copies the negative of the desired points in the input vector to the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vnmsaP

Vector Negative Multiply and Scalar Add

Function Definitions

```
void vnmsa(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vnmsa_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vnmsad(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vnmsad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rs1	input	Real scalar additive factor
rvo	output	Output vector
rvoinc	input	Increment for output vector
n	input	Number of points to move

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, negates that and adds real scalar rs1, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vnorm

Generate a normally-distributed vector

Function Definitions

```
void vnorm(
                    long  *pSeed,
                    float  mean,
                    float  dev,
                    float *v,
                    long   vinc,
                    long   n);

void vnorm_(
                    long  *pSeed,
                    float *mean,
                    float *dev,
                    float *v,
                    long  *vinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
pSeed	input	Pointer to random generator seed
mean	input	Mean value
dev	input	Standard deviation
v	output	Output vector
incv	input	Output vector increment
n	input	Number of samples to generate

Return Value

None: void function.

Description

This function generates a vector containing normally-distributed random values. The resulting vector has the specified mean and dev.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is only supported in single precision.

vnormz

Generate normally-distributed complex vector

Function Definitions

```
void vnormz(
                    long  *pSeed,
                    float *ctr,
                    float  dev,
                    float *v,
                    long   vinc,
                    long   n);

void vnormz_(
                    long  *pSeed,
                    float *ctr,
                    float *dev,
                    float *v,
                    long  *vinc,
                    long  *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
pSeed	input	Pointer to random generator seed
ctr	input	Complex "center" (mean value)
dev	input	Standard deviation
v	output	Output complex vector
incv	input	Output vector increment
n	input	Number of samples to generate

Return Value

None: void function.

Description

This function generates a vector containing normally-distributed complex random values. Resulting values are centered at *ctr, which is a point in the complex plane. Resulting values have a standard deviation of dev about that center.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is only supported in single precision.

volsave

Overlap and Save

Function Definitions

```
void volsave(
                    float *rv1,
                    long   rv1inc,
                    float *save,
                    long   slength,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void volsave_(
                    float *rv1,
                    long  *rv1inc,
                    float *save,
                    long  *slength,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
save	input	Address of real save buffer
slen	input	Length of save buffer
rvo	output	Real output vector
rvoinc	input	Output vector stride
n	input	length of input vector rv1

Return Value

None: void function.

Description

This function performs an overlap and save function:

- The first slen samples in the output vector are copied from the save buffer
- The next n samples of the output vector are copied from the input vector
- The last slen samples of the input buffer are copied into the save buffer

The resulting output vector is $slen + n$ elements long. The input vector length must be greater than or equal to slen.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.

vpmergP

Vector Positive Merge

Function Definitions

```
void vpmerg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vpmerg_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vpmergd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vpmergd_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Points from vectors rv1 and rv2 are transferred to vector rvo depending on the values in rv3. If an rv3 point is less than zero, the corresponding point in rv2 is transferred, otherwise the point is taken from rv1.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vpolyP

Vector Polynomial Evaluation

Function Definitions

```
void vpoly(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    long   n1);

void vpoly_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    long  *n1);

void vpolyd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    long    n1);

void vpolyd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    long   *n1);
```

Parameters

Argument	I/O	Description
rv1	input	Coefficient vector - highest order first
rv1inc	input	Vector increment for rv1
rv2	input	Input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process
n1	input	Order of polynomial (size of rv1-1)

Return Value

None: void function.

Description

Does an element by element polynomial evaluation using elements of rv1 as the coefficients to be applied to elements of rv2. Results are stored to rvo. The coefficients are stored in rv1 in descending order, starting with rv1[0] as the highest order coefficient. The number of stored coefficients is 1 plus the order of the polynomial.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vpythgP

Vector Pythagorean

Function Definitions

```
void vpythg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vpythg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vpythgd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vpythgd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Vector increment for rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment for rv2
`rv3`	input	Real input vector 3
`rv3inc`	input	Vector increment for rv3
`rv4`	input	Real input vector 4
`rv4inc`	input	Vector increment for rv4
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Computes the square root of the sum of the squares of the differences between the first and third input vectors, and the second and fourth input vectors.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vqintP

Vector Quadratic Interpolate

Function Definitions

```
void vqint(
                    float *rv1,
                    long   n1,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vqint_(
                    float *rv1,
                    long  *n1,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vqintd(
                    double *rv1,
                    long    n1,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vqintd_(
                    double *rv1,
                    long   *n1,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real compact input vector
n1	input	Integer size of vector rv1
rv2	input	Second real input vector
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

A quadratic interpolation is performed between selected points on input vector rv1 by using selected points of input vector rv2 as an index (whole number part), interpolation factor (fractional part), and factor squared (fractional part times itself). Results are put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vrampP

Generate a Ramp in a Vector

Function Definitions

```
void vramp(
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vramp_(
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vrampd(
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vrampd_(
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rs1	input	Initial ramp value
rs2	input	Ramp increment
rvo	output	Real output vector
rvoinc	input	Increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Generates a linear ramp starting with rs1 with successive points differing by rs2. The ramp is written to output vector rvo.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vrandP

Random Number Generator

Function Definitions

```
void vrand(
                    long  *is,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vrand_(
                    long  *is,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vrandd(
                    long   *is,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vrandd_(
                    long   *is,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
is	in/output	32-bit integer seed
rvo	output	Real output vector
rvoinc	input	Sample increment of rvo
n	input	Number of output samples

Return Value

None: void function.

Description

Uniformly distributed random numbers in the range of 0.0 to 0.999999 are generated. An updated 32-bit integer seed is returned after each call.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vrandz

Generate uniform random complex vector

Function Definitions

```
void vrandz(
                    long  *is,
                    long  *ctr,
                    long   dev,
                    float *v,
                    long   vinc,
                    long   n);

void vrandz_(
                    long  *is,
                    long  *ctr,
                    long  *dev,
                    float *v,
                    long  *vinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
pSeed	input	Pointer to random generator seed
ctr	input	Complex "center" (mean value)
dev	input	Standard deviation
v	output	Output complex vector
incv	input	Output vector increment
n	input	Number of samples to generate

Return Value

None: void function.

Description

This function generates a vector containing uniformly-distributed complex random values. Resulting values fall within a circle of radius dev centered at *ctr, which is a point in the complex plane.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

vrealP

Extract Real Part of Complex Vector

Function Definitions

```
void vreal(
                    float *cv1,
                    long   cv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vreal_(
                    float *cv1,
                    long  *cv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vreald(
                    double *cv1,
                    long    cv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vreald_(
                    double *cv1,
                    long   *cv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cv1	input	Complex input vector
cv1inc	input	Vector increment for cv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The real part of the complex input vector cv1 is copied to real output vector rvo.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex buffer is in Complex Interleaved format.

vrecipP

Compute Reciprocal of Vector

Function Definitions

```
void vrecip(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vrecip_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vrecipd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vrecipd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

The reciprocal of each point in the input vector is written to the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vrelerr

Relative Error

Function Definitions

```
void vrelerr(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long   n);

void vrelerr_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *avg,
                    float *max,
                    long  *i,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rv2	input	Input real vector
rv2inc	input	Input vector increment
avg	output	Average error
max	output	Maximum error
i	output	Index where max occurred
n	input	Number of samples to process

Return Value

None: void function.

Description

The absolute error computes fabs $[(rv1krv2k)/rv2k]$ for all k elements in each vector. The average of this value is computed and stored at *avg. The maximum error value is stored at *max, and the index to the maximum value is stored at *i.

Restrictions and Notes

1. The strided elements in the input vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

vround

Round

Function Definitions

```
void vround(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vround_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function rounds each input vector element to the nearest integer value. For example, round(2) = 2; round(-3.7) = -4, round (4.1) = 4.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in float only (no double precision).

vrsqrtP

Vector Reciprocal Square Root

Function Definitions

```
void vrsqrt(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vrsqrt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vrsqrtd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vrsqrtd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the reciprocal of the square root of each element of the real input vector and places the result into the real output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vrvrsP

Reverse a Vector

Function Definitions

```
void vrvrs(
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vrvrs_(
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vrvrsd(
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vrvrsd_(
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	in/output	Real vector to reverse
rv1inc	input	Increment for vector
n	input	Number of points to reverse

Return Value

None: void function.

Description

Reverses the order of the desired points in a vector. (Points towards the beginning of the vector are swapped with points towards the end of the vector.)

Restrictions and Notes

1. The strided elements in the input/output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input/output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsaddP

Add a Scalar to a Vector

Function Definitions

```
void vsadd(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsadd_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsaddd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsaddd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rs1	input	Real scalar to add
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Adds the scalar rs1 to the desired points in the input vector and stores the result in the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsaw

Generate Sawtooth Signal

Function Definitions

```
void vsaw(
                    float *freq,
                    float *amp,
                    float *phase,
                    float *v,
                    long   vinc,
                    long   n);

void vsaw_(
                    float *freq,
                    float *amp,
                    float *phase,
                    float *v,
                    long  *vinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
freq	input	Input frequency, in radians per sample
amp	input	Amplitude
phase	input	starting phase
v	output	Output real vector
incv	input	Output vector increment
n	input	Number of samples to generate

Return Value

None: void function.

Description

This function generates a real sawtooth signal having the specified amplitude and phase. Resulting values ramp from -amp to +amp and then drop back to -amp. phase determines the starting point of the signal and is typically zero.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is only supported in single precision.

vsbmP

Vector Subtract and Multiply

Function Definitions

```
void vsbm(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsbm_(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsbmd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsbmd_(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rv3	input	Real input vector 3
rv3inc	input	Vector increment for rv3
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Subtracts real input vectors rv2 from rv1, then multiplies by input vector rv3, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsbsbmP

Vector Subtract, Subtract, and Multiply

Function Definitions

```
void vsbsbm(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rv3,
                    long   rv3inc,
                    float *rv4,
                    long   rv4inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsbsbm_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rv3,
                    long  *rv3inc,
                    float *rv4,
                    long  *rv4inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsbsbmd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rv3,
                    long    rv3inc,
                    double *rv4,
                    long    rv4inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsbsbmd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rv3,
                    long   *rv3inc,
                    double *rv4,
                    long   *rv4inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real vector one
`rv1inc`	input	Increment vector one
`rv2`	input	Real vector two
`rv2inc`	input	Increment vector two
`rv3`	input	Real vector three
`rv3inc`	input	Increment vector three
`rv4`	input	Real vector four
`rv4inc`	input	Increment vector four
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

Takes the difference of real input vectors rv1 and rv2, then multiplies by the difference of input vectors rv3 and rv4, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsbsmP

Vector Subtract and Scalar Multiply

Function Definitions

```
void vsbsm(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsbsm_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsbsmd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsbsmd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real vector one
rv1inc	input	Increment for vector one
rv2	input	Real vector two
rv2inc	input	Increment for vector two
rs1	input	Real scalar multiplier
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Subtracts the input real vector rv2 from real vector rv1, then multiplies by real scalar rs1, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vscatr

Vector Scatter

Function Definitions

```
void vscatr(
                    float *rv1,
                    long   rv1inc,
                    float *iv2,
                    long   iv2inc,
                    float *rvo,
                    long   n);

void vscatr_(
                    float *rv1,
                    long  *rv1inc,
                    float *iv2,
                    long  *iv2inc,
                    float *rvo,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
iv2	input	Integer input vector
iv2inc	input	Vector increment for iv2
rvo	output	Real compact output vector
n	input	Number of points to output

Return Value

None: void function.

Description

Points from the input vector rv1 are placed into the output vector rvo at locations specified by the indices in the vector iv2. The index 0 specifies the first storage location in rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.

vsdivP

Divide Vector by Scalar

Function Definitions

```
void vsdiv(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsdiv_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsdivd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsdivd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rs1	input	Real divisor (non-zero)
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Divides the selected points in the input vector by the real scalar and stores the results to the output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsign

Signum Function

Function Definitions

```
void vsign(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsign_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the sign of each input vector element:

- Values < 0 result in a -1 result,
- values == 0 result in 0,
- values > = result in +1.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.

vsignz

Signum Function, Negative 0

Function Definitions

```
void vsignz(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsignz_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the sign of each input vector element:

- Values < 0 result in a -1 result,
- values == 0 result in 0,
- values > = result in +1.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.

vsimpsP

SimpsonâĂŹs Rule Integration

Function Definitions

```
void vsimps(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    float *rs1);

void vsimps_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    float *rs1);

void vsimpsd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    double *rs1);

void vsimpsd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    double *rs1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of points in output vector
rs1	input	The integration step size

Return Value

None: void function.

Description

vsimps integrates the vector rv1 with step size rs1 and stores the result in vector rvo, using SimpsonâĂŹs parabolic rule.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsinP

Vector Sine.

Function Definitions

```
void vsin(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsin_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsind(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsind_(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Compute the sine for each element of a vector. Element angle values are in radians.
rvo$[j * \texttt{rvoinc}]$:= sin(rv1$[j * \texttt{rv1inc}]$) where $0 \leqslant j < n$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.
6. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
7. Input arguments are expressed in radians.

vsinfP

Vector Sine in Fractions.

Function Definitions

```
void vsinf(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsinf_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsinfd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsinfd_(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the sine of each element of the real input vector and places the results in the real output vector. The elements of the input vector rv1 are assumed to be in fractions of a circle

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.
6. The input and output buffers must be identical (in-place), or must not overlap (out-of-place).
7. Input arguments are expressed in radians.

vsinrfP

Vector Sine in Fractions, Reduced Range.

Function Definitions

```
void vsinrf(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsinrf_(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsinrfd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsinrfd_(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

mputes the sine of each element of the real input vector and places the results in the real output vector. The elements of the input vector rv1 are assumed to be in fractions of a circle, limited to the range -0.25 to $+0.25$.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Accuracy is decreased for values larger than 8192.

vsm2saP

Multiply Two Vectors by Scalars and Add a Scalar

Function Definitions

```
void vsm2sa(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rv2,
                    long   rv2inc,
                    float *rs2,
                    float *rs3,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsm2sa_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rv2,
                    long   rv2inc,
                    float *rs2,
                    float *rs3,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsm2sad(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rv2,
                    long    rv2inc,
                    double *rs2,
                    double *rs3,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsm2sad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rv2,
                    long    rv2inc,
                    double *rs2,
                    double *rs3,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rv1inc`	input	Vector increment for rv1
`rs1`	input	Scalar to multiply with rv1
`rv2`	input	Real input vector 2
`rv2inc`	input	Vector increment for rv2
`rs2`	input	Scalar to multiply with rv2
`rs3`	input	Scalar to add
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of samples to process

Return Value

None: void function.

Description

For each point to be output, a sum is generated of: a point from the first input vector multiplied by its scalar, a point from the second input vector multiplied by its scalar, and finally the additive scalar. The result is written to the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsmaP

Vector Scalar Multiply and Add

Function Definitions

```
void vsma(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsma_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsmad(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsmad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rs1	input	Real scalar multiplier
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies the input real vector rv1 by real scalar rs1, adds input real vector rv2, and puts the results in real output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsma2P

Two Vector Multiply and Scalar Add

Function Definitions

```
void vsma2(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    long   inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsma2_(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    long  *inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsma2d(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    long    inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsma2d_(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    long   *inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rs1	input	Real scalar multiplicative factor
rv2	input	Real input vector 2
rs2	input	Real scalar multiplicative factor
inc	input	Vector increment for rv1 and rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, both with the same increment inc, by real scalar inputs rs1 and rs2, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsma2aP

Two Vector Multiply and Scalar Add, Accumulate

Function Definitions

```
void vsma2a(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    long   inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsma2a_(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    long  *inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsma2ad(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    long    inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsma2ad_(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    long   *inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rs1	input	Real scalar multiplicative factor
rv2	input	Real input vector 2
rs2	input	Real scalar multiplicative factor
inc	input	Vector increment for rv1 and rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1 and rv2, both with the same increment inc, by real scalar inputs rs1 and rs2, and adds the sum to output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsma3P

Three Vector Multiply and Scalar Add

Function Definitions

```
void vsma3(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    float *rv3,
                    float *rs3,
                    long   inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsma3_(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    float *rv3,
                    float *rs3,
                    long  *inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsma3d(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    double *rv3,
                    double *rs3,
                    long    inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsma3d_(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    double *rv3,
                    double *rs3,
                    long   *inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
`rv1`	input	Real input vector 1
`rs1`	input	Multiplier for rv1
`rv2`	input	Real input vector 2
`rs2`	input	Multiplier for rv2
`rv3`	input	Real input vector 3
`rs3`	input	Multiplier for rv3
`inc`	input	Increment for rv1, rv2, rv3
`rvo`	output	Real output vector
`rvoinc`	input	Increment for output vector
`n`	input	Number of points to move

Return Value

None: void function.

Description

Multiplies real input vectors rv1, rv2, and rv3, all with the same increment inc, by real scalar inputs rs1, rs2, and rs3, respectively, and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsma4P

Four Vector Multiply and Scalar Add

Function Definitions

```
void vsma4(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    float *rv3,
                    float *rs3,
                    float *rv4,
                    float *rs4,
                    long   inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsma4_(
                    float *rv1,
                    float *rs1,
                    float *rv2,
                    float *rs2,
                    float *rv3,
                    float *rs3,
                    float *rv4,
                    float *rs4,
                    long  *inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsma4d(
                    double *rv1,
                    double *rs1,
                    double *rv2,
                    double *rs2,
                    double *rv3,
                    double *rs3,
                    double *rv4,
                    double *rs4,
                    long    inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);
```

```
void vsma4d_(
                        double *rv1,
                        double *rs1,
                        double *rv2,
                        double *rs2,
                        double *rv3,
                        double *rs3,
                        double *rv4,
                        double *rs4,
                        long   *inc,
                        double *rvo,
                        long   *rvoinc,
                        long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rs1	input	Multiplier for rv1
rv2	input	Real input vector 2
rs2	input	Multiplier for rv2
rv3	input	Real input vector 3
rs3	input	Multiplier for rv3
rv4	input	Real input vector 4
rs4	input	Multiplier for rv4
inc	input	Increment for rv1, rv2, rv3, rv4
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies real input vectors rv1, rv2, rv3, and rv4, all with same increment inc, by real scalar inputs rs1, rs2, rs3, and rv4 respectively, and puts the result in output real vector rvo for n points

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsmsaP

Multiply Vector by a Scalar and Add Scalar

Function Definitions

```
void vsmsa(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsmsa_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsmsad(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsmsad_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rs1	input	Scalar to multiply
rs2	input	Scalar to add
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Each selected point in the input vector is multiplied by the real scalar and then has the real scalar added to it. The result is stored in the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsmsbP

Vector Scalar Multiply and Subtract

Function Definitions

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rs1	input	Real scalar multiplier
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies the input real vector rv1 by real scalar rs1, subtracting input real vector rv2, and putting the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsmulP

Multiply Vector by a Scalar

Function Definitions

```
void vsmul(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsmul_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsmuld(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsmuld_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rs1	input	Scalar to multiply
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies the selected points in the input vector by the real scalar and stores the result in the output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsort

Vector Sort

Function Definitions

```
void vsort(
                    float *rv1,
                    float *rvo,
                    long   n,
                    long   order);

void vsort_(
                    float *rv1,
                    float *rvo,
                    long  *n,
                    long  *order);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rvo	output	Output real vector
n	input	Number of samples to process
order	input	Sorting order

Return Value

None: void function.

Description

If order < 0, the vector is sorted in descending order. Otherwise, the vector is sorted in ascending order.

Restrictions and Notes

1. Both the input and output vectors are in compact format.
2. The input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when the **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.

vspdp

Vector Convert Single to Double Precision

Function Definitions

```
void vspdp(
                    float  *rv1,
                    long    rv1inc,
                    double *dvo,
                    long    dvoinc,
                    long    n);

void vspdp_(
                    float  *rv1,
                    long   *rv1inc,
                    double *dvo,
                    long   *dvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
dvo	output	Double precision output vector
dvoinc	input	Increment for output vector
n	input	Number of points to convert

Return Value

None: void function.

Description

Converts the input real vector rv1 to double precision and puts the result in output double precision vector dvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.

vsqP

Vector Square

Function Definitions

```
void vsq(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsqd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsqd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Squares real input vector rv1 and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsqrtP

Vector Square Root

Function Definitions

```
void vsqrt(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsqrt_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsqrtd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsqrtd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the square root of each element of the real input vector and places the result into the real output vector.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsquare

Generate Square Wave Signal

Function Definitions

```
void vsquare(
                    float  freq,
                    float  amp,
                    float  phase,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsquare_(
                    float *freq,
                    float *amp,
                    float *phase,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
freq	input	Input frequency, in radians per sample
amp	input	Amplitude
phase	input	starting phase
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to generate

Return Value

None: void function.

Description

This function generates a real square wave signal having the specified amplitude and phase. Resulting values are $+amp$ or $-amp$. phase determines the starting point of the signal and is typically zero.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is only supported in single precision.

vssqP

Vector Signed Square

Function Definitions

```
void vssq(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vssq_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vssqd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vssqd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Multiplies the real input vector rv1 with its absolute value and puts the result in output real vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsubP

Subtract One Vector from Another

Function Definitions

```
void vsub(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vsub_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vsubd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vsubd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Subtracts vector rv1âĂŹs points from vector rv2 and stores the results in the output vector rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vsumP

Vector Sum

Function Definitions

```
void vsum(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    float *rs1);

void vsum_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    float *rs1);

void vsumd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    double *rs1);

void vsumd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    double *rs1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Vector increment for output vector
n	input	Number of points in output vector
rs1	input	Integration step size

Return Value

None: void function.

Description

The values in vector rv1 are integrated using the step size rs1. The resulting integral vector is stored into rvo.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vswapP

Vector Swap

Function Definitions

```
void vswap(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    long   n);

void vswap_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    long  *n);

void vswapd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    long    n);

void vswapd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	in/output	Real vector 1
rv1inc	input	Vector increment for rv1
rv2	in/output	Real vector 2
rv2inc	input	Vector increment for rv2
n	input	Number of points to swap

Return Value

None: void function.

Description

The elements of vectors rv1 and rv2 are swapped with each other.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vtabiP

Vector Table Look-up, Linear Interpolate

Function Definitions

```
void vtabi(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rv2,
                    float *rvo,
                    long   rvoinc,
                    long   n2,
                    long   n);

void vtabi_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rs2,
                    float *rv2,
                    float *rvo,
                    long  *rvoinc,
                    long  *n2,
                    long  *n);

void vtabid(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rv2,
                    double *rvo,
                    long    rvoinc,
                    long    n2,
                    long    n);

void vtabid_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rs2,
                    double *rv2,
                    double *rvo,
                    long   *rvoinc,
                    long   *n2,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Integer increment for rv1
rs1	input	Real multiplicative scalar
rs2	input	Real additive scalar
rv2	input	Second real input vector (table)
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n2	input	Integer size of vector (table)
n	input	Number of samples to process

Return Value

None: void function.

Description

After scaling points in rv1, table look-up and linear interpolation are used with table rv2 to determine the value that will be written into rvo.

Restrictions and Notes

1. The strided elements in the output vector must be equal to or greater than **n** in length.
2. The strided elements in the input vector must be equal to or greater than **n2** in length.
3. The operation is performed faster when **n** and **n2** are set to a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.

vtanP

Vector Tangent

Function Definitions

```
void vtan(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vtan_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vtand(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vtand_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the tangent of each element of the real input vector and places the results in the real output vector. The elements of the input vector are assumed to be in radians.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vtanfP

Vector Tangent in Fractions of a Circle

Function Definitions

```
void vtanf(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vtanf_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vtanfd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vtanfd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Increment for input vector
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Computes the tangent of each element in the real input vector and places the result in the real output vector. The elements of the input vector are assumed to be in fractions of a circle.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vthrP

Vector Threshold

Function Definitions

```
void vthr(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vthr_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vthrd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vthrd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rs1	input	Real scalar threshold
rvo	output	Real vector output
rvoinc	output	Vector increment for rvo
n	input	Number of samples to process

Return Value

None: void function.

Description

Compare vector points against threshold, take larger value and put into output vector rvo for n points.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vthresP

Replace Elements Less Than Scalar with Zero

Function Definitions

```
void vthres(
                    float *rv1,
                    long   rv1inc,
                    float *rs1,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vthres_(
                    float *rv1,
                    long  *rv1inc,
                    float *rs1,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vthresd(
                    double *rv1,
                    long    rv1inc,
                    double *rs1,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vthresd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rs1,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rs1	input	Real scalar minimum value
rvo	output	Real vector output
rvoinc	input	Vector increment for rvo
n	input	Number of points in output vector

Return Value

None: void function.

Description

Copies elements from rv1 into rvo, substituting 0 if the element from rv1 is less than scalar rs1.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vtmergP

Vector Tapered Merge

Function Definitions

```
void vtmerg(
                    float *rv1,
                    long   rv1inc,
                    float *rv2,
                    long   rv2inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vtmerg_(
                    float *rv1,
                    long  *rv1inc,
                    float *rv2,
                    long  *rv2inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);

void vtmergd(
                    double *rv1,
                    long    rv1inc,
                    double *rv2,
                    long    rv2inc,
                    double *rvo,
                    long    rvoinc,
                    long    n);

void vtmergd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rv2,
                    long   *rv2inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector 1
rv1inc	input	Vector increment for rv1
rv2	input	Real input vector 2
rv2inc	input	Vector increment for rv2
rvo	output	Real output vector
rvoinc	input	Increment for output vector
n	input	Number of samples to process

Return Value

None: void function.

Description

Input vectors rv1 and rv2 are merged into vector rvo by adding a fraction "f" of each element in rv1 to the fraction 1.0-f of each element of rv2 to form elements of rvo. "f" varies linearly from 1.0 at the first element to 0.0 at the last.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vtrapzP

Trapezoidal Rule Integration

Function Definitions

```
void vtrapz(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n,
                    float *rs1);

void vtrapz_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n,
                    float *rs1);

void vtrapzd(
                    double *rv1,
                    long    rv1inc,
                    double *rvo,
                    long    rvoinc,
                    long    n,
                    double *rs1);

void vtrapzd_(
                    double *rv1,
                    long   *rv1inc,
                    double *rvo,
                    long   *rvoinc,
                    long   *n,
                    double *rs1);
```

Parameters

Argument	I/O	Description
rv1	input	Real input vector
rv1inc	input	Vector increment for rv1
rvo	output	Real output vector
rvoinc	input	Output vector increment
n	input	Number of samples to process
rs1	input	The integration step size

Return Value

None: void function.

Description

vtrapz integrates the vector rv1 with step size rs1 and stores the result in vector rvo, using the trapezoidal rule for integration.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is supported in both single and double precision.

vtrunc

Truncate

Function Definitions

```
void vtrunc(
                    float *rv1,
                    long   rv1inc,
                    float *rvo,
                    long   rvoinc,
                    long   n);

void vtrunc_(
                    float *rv1,
                    long  *rv1inc,
                    float *rvo,
                    long  *rvoinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
rv1	input	Input real vector
rv1inc	input	Input vector increment
rvo	output	Output real vector
rvoinc	input	Output vector increment
n	input	Number of samples to process

Return Value

None: void function.

Description

This function truncates each to an integer value. The truncation function discards the fractional portion of each value. For example, trunc(2) = 2; trunc (-3.1) = -3, trunc (4.1) = 4.

Restrictions and Notes

1. The strided elements in the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when the **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.

vxcsP

Real Vector Multiplied by Complex Exponential

Function Definitions

```
void vxcs(
                    float *rv1,
                    float *cvo,
                    long   cvoinc,
                    float *rs1,
                    float *rs2,
                    long   n);

void vxcs_(
                    float *rv1,
                    float *cvo,
                    long  *cvoinc,
                    float *rs1,
                    float *rs2,
                    long  *n);

void vxcsd(
                    double *rv1,
                    double *cvo,
                    long    cvoinc,
                    double *rs1,
                    double *rs2,
                    long    n);

void vxcsd_(
                    double *rv1,
                    double *cvo,
                    long   *cvoinc,
                    double *rs1,
                    double *rs2,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
rv1	input	Real input vector
cvo	output	Complex output vector
cvoinc	input	Vector increment for cvo
rs1	input	Frequency
rs2	in/output	Phase
n	input	Number of points in output vector

Return Value

None: void function.

Description

A complex exponential is formed from the initial phase rs2 and frequency rs1 (in units of radians). The real part of each element of the complex exponential is the cosine of the current phase step; the imaginary part is the sine of the current phase step. The next phase step that would follow the final phase used is written to the scalar rs2. Each element of the input vector rv1 is multiplied by the current complex exponential to form the complex output vector cvo.

Restrictions and Notes

1. The strided elements in output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex output buffer is in Complex Interleaved format.

vxcsfP

Real Vector Multiplied by Fractional Complex Exponential

Function Definitions

```
void vxcsf(
                    float *rv1,
                    long   rv1inc,
                    float *cvo,
                    long   cvoinc,
                    float *rs1,
                    float *rs2,
                    long   n);

void vxcsf_(
                    float *rv1,
                    long  *rv1inc,
                    float *cvo,
                    long  *cvoinc,
                    float *rs1,
                    float *rs2,
                    long  *n);

void vxcsfd(
                    double *rv1,
                    long    rv1inc,
                    double *cvo,
                    long    cvoinc,
                    double *rs1,
                    double *rs2,
                    long    n);

void vxcsfd_(
                    double *rv1,
                    long   *rv1inc,
                    double *cvo,
                    long   *cvoinc,
                    double *rs1,
                    double *rs2,
                    long   *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
`rv1`	input	Complex input vector
`rv1inc`	input	Vector increment for rv1
`cvo`	output	Complex output vector
`cvoinc`	input	Vector increment for rvo
`rs1`	input	Frequency
`rs2`	in/output	Phase
`n`	input	Number of samples to process

Return Value

None: void function.

Description

A complex exponential is formed from the initial phase rs2 and frequency rs1 (in units of fractions of a circle). The real part of each element of the complex exponential is the cosine of the current phase step; the imaginary part is the sine of the current phase step. The next phase step that follows the final phase used is written to the scalar rs2. Each element of the input vector rv1 is multiplied by the current complex exponential to form the complex output vector cvo.

Restrictions and Notes

1. The strided elements in output vector must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the output vector is fully aligned for the target hardware.
4. This operation is supported in both single and double precision.
5. Complex output buffer is in Complex Interleaved format.

wienerP

Wiener Levinson Equation Solver

Function Definitions

```
void wiener(
                    long   LR,
                    float *R,
                    float *G,
                    float *F,
                    float *A,
                    long   ISW,
                    long  *IERR);

void wiener_(
                    long  *LR,
                    float *R,
                    float *G,
                    float *F,
                    float *A,
                    long  *ISW,
                    long  *IERR);

void wienerd(
                    long    LR,
                    double *R,
                    double *G,
                    double *F,
                    double *A,
                    long    ISW,
                    long   *IERR);

void wienerd_(
                    long   *LR,
                    double *R,
                    double *G,
                    double *F,
                    double *A,
                    long   *ISW,
                    long   *IERR);
```

Parameters

Argument	I/O	Description
LR	input	Length of the filter
R	input	Real vector with Auto-correlation coefficients
G	input	Real vector with Cross-correlation coefficients
ISW	input	Mode selector switch ISW ne 0: General deconvolution ISW = 0: Spike deconvolution.
F	output	Real array of filter coefficients.
A	output	Real array of coefficients of the prediction error operator.
IERR	output	Scalar Error flag. IERR = 0 The routine completed without error. IERR = -1 Invalid parameter, $LR < 1$. IERR = L The routine encountered a division by zero on the L-th pass of algorithm.

Return Value

None: void function.

Description

To solve a system of single channel normal equations which arise in least-squares filtering and prediction problems.

Restrictions and Notes

1. Input and output vectors are of compact format.
2. The input and output vectors must be equal to or greater than **LR** in length.
3. The operation is performed faster when **LR** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is supported in both single and double precision.

zadd

Split Complex Add

Function Definitions

```
void zadd(
                    float         *cvxr,
                    float         *cvxi,
                    long           cvxinc,
                    float         *cvyr,
                    float         *cvyi,
                    long           cvyinc,
                    float         *cvzr,
                    float         *cvzi,
                    long           cvzinc,
                    unsigned long  n);

void zadd_(
                    float         *cvxr,
                    float         *cvxi,
                    long          *cvxinc,
                    float         *cvyr,
                    float         *cvyi,
                    long          *cvyinc,
                    float         *cvzr,
                    float         *cvzi,
                    long          *cvzinc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex Input vector, real components
cvxi	input	Split complex Input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex Input vector, real
cvyi	input	Split complex Input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function adds the complex split vector **(cvxr, cvxi)** to the complex split vector **(cvyr,cvyi)** giving the complex split vector **(cvzr, cvzi)**.

Restrictions and Notes

1. All vectors are strided vectors with a required element increment.
2. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. All complex vectors are in Complex Split format.

zconj

Split Complex Conjugate

Function Definitions

```
void zconj(
                    float         *cvxr,
                    float         *cvxi,
                    long           cvxinc,
                    float         *cvzr,
                    float         *cvzi,
                    long           cvzinc,
                    unsigned long  n);

void zconj_(
                    float         *cvxr,
                    float         *cvxi,
                    long          *cvxinc,
                    float         *cvzr,
                    float         *cvzi,
                    long          *cvzinc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function copies the complex split vector **(cvxr, cvxi)** to complex split vector **(cvzr, cvzi)**, negating the imaginary component.

Restrictions and Notes

1. All vectors are strided vectors with a required element increment.
2. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. All complex vectors are in Complex Split format.

zconv

Split Complex Convolution

Function Definitions

```
void zconv(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long  nz,
                    unsigned long  ny);

void zconv_(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long *nz,
                    unsigned long *ny);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvyr	input	Split complex input correlation pattern, real components
cvyi	input	Split complex input correlation pattern, imaginary components
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
nz	input	Number of samples to process
ny	input	number of samples in vector y

Return Value

None: void function.

Description

This computes the convolution between the input vector cvx and the convolution kernel cvy. Note that the vector cvy should be reversed prior to calling this function in order to compute the convolution. The kernel (cvy vector) is ny elements in length. The input vector is assumed to be nz + ny - 1 elements in length.

Restrictions and Notes

1. All vectors are in compact format.
2. The the input vector **cvx** is equal to or greater than **nz+ny-1** in length.
3. The the input vector **cvy** is equal to or greater than **ny** in length.
4. The the output vector **cvz** is equal to or greater than **nz** in length.
5. The operation is performed faster when **ny** and **nz** are a power of two.
6. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
7. This operation is only supported in single precision.
8. All complex vectors are in Complex Split format.

zconvd

Decimating Convolution

Function Definitions

```
void zconvd(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long  nz,
                    unsigned long  ny,
                    unsigned long  ndf);
```

```
void zconvd_(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long *nz,
                    unsigned long *ny,
                    unsigned long *ndf);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvyr	input	Split complex input correlation pattern, real components
cvyi	input	Split complex input correlation pattern, imaginary components
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
nz	input	number of samples to process
ny	input	number of samples in vector y
ndf	input	decimation factor

Return Value

None: void function.

Description

This function is similar to zconv, except after computing each output point, the input pointer to the x vector is incremented by a step size of nfd points. This function can be used to provide a complex decimating finite impulse response (FIR) filter.

Restrictions and Notes

1. All vectors are in compact format.
2. The the input vector **x** is equal to or greater than **nz+ny-1** in length.
3. The the input vector **y** is equal to or greater than **ny** in length.
4. The the output vector **z** is equal to or greater than **nz** in length.
5. The operation is performed faster when **ny** and **nz** are a power of two.
6. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
7. This operation is only supported in single precision.
8. All complex vectors are in Complex Split format.

zcorr

Split Complex Correlation

Function Definitions

```
void zcorr(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long  nz,
                    unsigned long  ny);

void zcorr_(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvyr,
                    float         *cvyi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long *nz,
                    unsigned long *ny);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvyr	input	Split complex input correlation pattern, real components
cvyi	input	Split complex input correlation pattern, imaginary components
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
nz	input	Number of samples to process
ny	input	number of samples in vector y

Return Value

None: void function.

Description

This computes the time domain cross-correlation between the input vector x and the correlation pattern y. Both the input vector and the pattern are ny elements in length. The result is never more than $2 * ny - 1$ points in length.

Restrictions and Notes

1. All vectors are in compact format.
2. The the input vector **x** is equal to or greater than **ny** in length.
3. The the input vector **y** is equal to or greater than **ny** in length.
4. The the output vector **z** is equal to or greater than **nz** in length.
5. The operation is performed faster when **ny** and **nz** are a power of two.
6. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
7. This operation is only supported in single precision.
8. All complex vectors are in Complex Split format.

zdiv

Split Complex Divide

Function Definitions

```
void zdiv(
                    float         *cvxr,
                    float         *cvxi,
                    long           cvxinc,
                    float         *cvyr,
                    float         *cvyi,
                    long           cvyinc,
                    float         *cvzr,
                    float         *cvzi,
                    long           cvzinc,
                    unsigned long  n);

void zdiv_(
                    float         *cvxr,
                    float         *cvxi,
                    long          *cvxinc,
                    float         *cvyr,
                    float         *cvyi,
                    long          *cvyinc,
                    float         *cvzr,
                    float         *cvzi,
                    long          *cvzinc,
                    unsigned long *n);
```

Interleaved Complex Datatypes.

The definition of the Interleaved Complex datatype varies with which Vector include files a user includes. The Interleaved Complex datatypes are given in the table below:

Def No.	Include File	Complex Precision	Interleaved Complex Numbers Declared As...
1.	CV.h CVD.h	Single Double	float* array; (as given above) double* array; (as given above)
2.	CVC.h CVDC.h	Single Double	float array[][2]; double array[][2];
3.	CVSC.h CVDSC.h	Single Double	CV_COMPLEX* array; CV_DOUBLE_COMPLEX* array;

All complex arguments in the above function definitions are defind as Def 1. To use 2 or 3 replace each complex argument definition with one from the above table.

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	complex Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function divides the complex split vector (xr, xi) by the complex de-interleave vector (yr,yi) giving the complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zdot

Split Complex Dot Product

Function Definitions

```
void zdot(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   n);

void zdot_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output scalar, real
cvzi	output	Split complex output scalar, imaginary
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the dot product of the complex vectors x, y. Note that the complex dot product is, by definition, computed using the conjugate of the y vector.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zdotconj

Split Complex Dot Product Conjugate

Function Definitions

```
void zdotconj(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   n);

void zdotconj_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output scalar, real
cvzi	output	Split complex output scalar, imaginary
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the dot product of the complex vectors x, y. Note that the complex dot product is, by definition, computed using the conjugate of the y vector. This function computes the complex dot product using $conj(y)$, which is equivalent to using the true value of y, since the dot product itself implies conjugation.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zexp

Split Complex Exponentiate

Function Definitions

```
void zexp(
                    float         *cvxr,
                    float         *cvxi,
                    unsigned long  cvxinc,
                    float         *cvzr,
                    float         *cvzi,
                    long           cvzinc,
                    unsigned long  n);

void zexp_(
                    float         *cvxr,
                    float         *cvxi,
                    long          *cvxinc,
                    float         *cvzr,
                    float         *cvzi,
                    long          *cvzinc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

Description

This computes z = exp(x), where x and z are complex split vectors.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zfft

Split Complex to Complex FFT

Function Definitions

```
void zfft(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long  n);

void zfft_(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvzr,
                    float         *cvzi,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
n	input	Number of samples to process

Return Value

- This function returns 0 for success. A nonzero return code indicates one of the following errors:
  - -1: Input vector length is not supported
  - -2: Cannot allocate memory for weights
  - -3: Operands are not quad-word aligned
  - -4: Invalid parameter
  - -5: In-place not supported for this length

Description

This computes the complex FFT of the complex input vector x. This function does not support a real input; the xi pointer must be non-null.

Restrictions and Notes

1. All vectors are in compact format.
2. The input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. All complex vectors are in Complex Split format.
7. The variable **n** is the FFT length.

zfftfree

Split Deallocate FFT Weights

Function Definitions

```
void zfftfree(void);
```

Parameters

Argument	I/O	Description

Return Value

None: void function.

Description

This function frees all memory allocated by the FFT and zfftinit functions.

Restrictions and Notes

None.

zfftinit

Split Complex FFT Weights

Function Definitions

Parameters

Argument	I/O	Description

Return Value

- A return value of 0 indicates success. A nonzero value indicates an unsupported length, or insufficient memory.

Description

This allocates memory and computes complex weights needed to compute an FFT of length n.

Restrictions and Notes

1. The FFT operation performs faster if the FFT length is a power of 2.
2. The FFT operation performs faster if the input and output buffers are fully aligned for the target hardware.

zfill

Split Complex Fill

Function Definitions

```
void zfill(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvzr,
                    float         *cvzi,
                    long           cvzinc,
                    unsigned long  n);

void zfill_(
                    float         *cvxr,
                    float         *cvxi,
                    float         *cvzr,
                    float         *cvzi,
                    long          *cvzinc,
                    unsigned long *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This computes fills the complex split vector z with (xr, xi).

Restrictions and Notes

1. All vectors have a stride of incz.
2. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. All complex vectors are in Complex Split format.

zfloat16

Split Complex Float

Function Definitions

```
void zfloat16(
                    short *ivx,
                    float  rs1,
                    float *cvzr,
                    float *cvzi,
                    long   n);

void zfloat16_(
                    short *ivx,
                    float *rs1,
                    float *cvzr,
                    float *cvzi,
                    long  *n);
```

Parameters

Argument	I/O	Description
ivx	input	Short interleaved complex Vector
rs1	input	scaling value
cvzr	output	Split complex output real vector
cvzi	output	Split complex output imaginary vector
n	input	number of elements to process

Return Value

None: void function.

Description

This function converts complex interleaved 16 bit signed integer values to a split floating point vector. Vector elements are multiplied by scale during conversion.

Restrictions and Notes

1. All vectors are of type compact.
2. The input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. The output complex vector is in Complex Split format.
7. The input complex vector is in Complex Interleaved format.

zifft

Split Complex to Complex Inverse FFT

Function Definitions

Parameters

Argument	I/O	Description
`cvxr`	input	Split complex input vector, real components
`cvxi`	input	Split complex input vector, imaginary components
`cvzr`	output	Split complex output vector, real
`cvzi`	output	Split complex output vector, imaginary components
`n`	input	Number of samples to process

Return Value

- This function returns 0 for success. A nonzero return code indicates one of the following errors:
  - -1: Input vector length is not supported
  - -2: Cannot allocate memory for weights
  - -3: Operands are not quad-word aligned
  - -4: Invalid parameter
  - -5: In-place not supported for this length

Description

This computes the complex inverse FFT of the complex input vector x. This function does not support a real transforms; the input and output must both be complex.

Restrictions and Notes

1. All vectors are in compact format.
2. The input and output vectors must be equal to or greater than **n** in length.
3. The operation is performed faster when **n** is a power of two.
4. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
5. This operation is only supported in single precision.
6. All complex vectors are in Complex Split format.
7. The variable **n** is the FFT length.

zmadd

Split Complex Matrix Add

Function Definitions

```
void zmadd(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *cmyr,
                    float *cmyi,
                    long   yrows,
                    long   ycols,
                    float *cmzr,
                    float *cmzi);

void zmadd_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *cmyr,
                    float *cmyi,
                    long  *yrows,
                    long  *ycols,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cmyr	input	Split complex input matrix, real
cmyi	input	Split complex input matrix, imaginary components
yrows	input	Number of rows in matrix y
ycols	input	Number of columns in matrix y
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

- This function returns 0 if the matrices conform for addition (i.e., xcols = ycols and xrows = yrows), otherwise this function returns a nonzero value.

Description

This function adds the complex split matrix (xr, xi) to the complex de-interleave matrix (yr, yi) giving the complex split matrix (zr, zi). The resulting matrix has dimensions xrows by xcols.

Restrictions and Notes

1. The function will return a non-zero value if an error occurs.
2. The operation is performed faster when the matrix row column length is a power of two.
3. All input and output matrices are of type compact.
4. All input and output matrices are of type Split Complex.
5. The operation is performed faster when the input and output matrices are fully aligned for the target hardware.

6. This operation is only supported in single precision.

zmag

Magnitude of Split Complex Vector

Function Definitions

```
void zmag(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *rvz,
                    long   rvzinc,
                    long   n);

void zdiv_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *rvz,
                    long  *rvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
rvz	input	output vector, real
rvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the magnitude of all elements in the complex split vector (xr, xi). The resulting real vector is placed at yr.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zmag2

Magnitude Squared of Split Complex Vector

Function Definitions

```
void zmag2(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *rvz,
                    long   rvzinc,
                    long   n);

void zmag2_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *rvz,
                    long  *rvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
rvzy	input	output vector, real
rvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the magnitude squared of all elements in the complex split vector (xr, xi). The resulting real vector is placed at yr.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zmgetcol

Get a Column from a Split Complex Matrix.

Function Definitions

```
void zmgetcol(
                    float         *cmxr,
                    float         *cmxi,
                    long           xrows,
                    long           xcols,
                    float         *cvzr,
                    float         *cvzr,
                    long           cvzinc,
                    unsigned long  col);

void zmgetcol_(
                    float         *cmxr,
                    float         *cmxi,
                    long          *xrows,
                    long          *xcols,
                    float         *cvzr,
                    float         *cvzr,
                    long          *cvzinc,
                    unsigned long *col);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary
cvzinc	input	Vector increment for vectors zr and zi
col	input	which column to get

Return Value

None: void function.

Description

This function extracts the given column from the matrix split matrix $(\mathbf{xr}, \mathbf{xi})$ giving the vector $(\mathbf{zr}, \mathbf{zi})$. The resulting vector has length **xrows**.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column length is a power of two.
2. The input matrix is of type compact.
3. All input matrix is of type Split Complex.
4. The operation is performed faster when the input matrix and output vector are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmmov

Split Complex Sub-Matrix Move

Function Definitions

```
void zmmov(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    long   xstride,
                    float *cmzr,
                    float *cmzi,
                    long   zstride);

void zmmov_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    long  *xstride,
                    float *cmzr,
                    float *cmzi,
                    long  *zstride);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows to move from matrix x
xcols	input	Number of columns to move from matrix x
xstride	input	Row stride for matrix X: the actual number of columns in each row of x.
cmzr	output	output matrix, real
cmzi	output	output matrix, imaginary components
zstride	output	Row stride for matrix Z: the actual number of columns in each row of z.

Return Value

None: void function.

Description

This function moves a real or complex matrix or sub-matrix from x to z. The pointers xr, xi and zr, zi should point to the "upper left" element of a submatrix of x or z. The sub-matrix has dimensions xrows by xcols. Matrix x has rows containing xstr elements per row, and matrix z has zstr elements per row.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. The rows in the input matrix are of type compact.
3. The rows in the output matrix are of type compact.
4. All matrices are of type Split Complex.
5. The operation is performed faster when the input matrix and output vector are fully aligned for the target hardware.
6. This operation is only supported in single precision.

zmmult

Split Complex Matrix Multiply

Function Definitions

```
void zmmult(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *cmyr,
                    float *cmyi,
                    long   yrows,
                    long   ycols,
                    float *cmzr,
                    float *cmzi);

void zmmult_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *cmyr,
                    float *cmyi,
                    long  *yrows,
                    long  *ycols,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cmyr	input	Split complex input matrix, real
cmyi	input	Split complex input matrix, imaginary components
yrows	input	Number of rows in matrix y
ycols	input	Number of columns in matrix y
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

None: void function.

Description

This function multiplies the complex split matrix (xr, xi) by the complex deinterleave matrix (yr, yi) giving the complex split matrix (zr, zi). Note that the normal definition of complex matrix multiplication employs the conjugate of the 2nd operand. The resulting matrix has dimensions xrows by ycols.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All matrices are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmmults

Split Complex Matrix Multiply by Scalar

Function Definitions

```
void zmmults(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *csyr,
                    float *csyi,
                    float *cmzr,
                    float *cmzi);

void zmmults_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *csyr,
                    float *csyi,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
csyr	input	Split complex input scalar, real
csyi	input	Split complex input scalar, imaginary
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

None: void function.

Description

This function multiplies the complex split matrix $(\mathbf{xr}, \mathbf{xi})$ by the complex scalar $(\mathbf{yr}, \mathbf{yi})$ giving the complex split matrix $(\mathbf{zr}, \mathbf{zi})$. The resulting matrix has dimensions xrows by **xcols**.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All matrices are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmmulttr

Split Complex Matrix Multiply Transpose

Function Definitions

```
void zmmulttr(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *cmyr,
                    float *cmyi,
                    long   yrows,
                    long   ycols,
                    float *cmzr,
                    float *cmzi);

void zmmulttr_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *cmyr,
                    float *cmyi,
                    long  *yrows,
                    long  *ycols,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cmyr	input	Split complex input matrix, real
cmyi	input	Split complex input matrix, imaginary components
yrows	input	Number of rows in matrix y
ycols	input	Number of columns in matrix y
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

None: void function.

Description

This function multiplies the complex split matrix $(\mathbf{xr}, \mathbf{xi})$ by the transpose of the complex split matrix $(\mathbf{yr}, \mathbf{yi})$ giving the complex split matrix $(\mathbf{zr}, \mathbf{zi})$. Note that the normal definition of complex matrix multiplication employs the conjugate of the 2nd operand. The resulting matrix has dimensions **xrows** by **yrows**.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All matrices are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmmulttrconj

Split Complex Matrix Multiply Transpose Conjugate

Function Definitions

```
void zmmulttrconj(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *cmyr,
                    float *cmyi,
                    long   yrows,
                    long   ycols,
                    float *cmzr,
                    float *cmzi);

void zmmulttrconj_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *cmyr,
                    float *cmyi,
                    long  *yrows,
                    long  *ycols,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cmyr	input	Split complex input matrix, real
cmyi	input	Split complex input matrix, imaginary components
yrows	input	Number of rows in matrix y
ycols	input	Number of columns in matrix y
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

None: void function.

Description

This function multiplies the complex split matrix $(\mathbf{xr}, \mathbf{xi})$ by the transpose of the complex conjugate of the de-interleave matrix $(\mathbf{yr}, \mathbf{yi})$ giving the complex split matrix $(\mathbf{zr}, \mathbf{zi})$. Note that the normal definition of complex matrix multiplication employs the conjugate of the 2nd operand. Since these functions conjugate the y value, and since conj $(conj(y)) == y$, these functions use the "true" value of y. The resulting matrix has dimensions **xrows** by **yrows**.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All matrices are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.

5. This operation is only supported in single precision.

zmov

Split Complex Move

Function Definitions

```
void zmov(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zmov_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function copies the complex split vector (xr, xi) to complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zmsetcol

Set a Matrix Column

Function Definitions

```
void zmmulttrconj(
                    float *cvxr,
                    float *cvxi,
                    float *cmzr,
                    float *cmzi,
                    long   zrows,
                    long   zcols,
                    long   col);

void zmmulttrconj_(
                    float *cvxr,
                    float *cvxi,
                    float *cmzr,
                    float *cmzi,
                    long  *zrows,
                    long  *zcols,
                    long  *col);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cmzr	input	Split complex output matrix, real
cmzi	input	Split complex output matrix, imaginary
zrows	input	Number of rows in matrix z
zcols	input	Number of columns in matrix z
col	input	which column to set

Return Value

None: void function.

Description

This function copies elements from the row vector x into column col of matrix z.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column length is a power of two.
2. The output matrix is of type compact.
3. All buffers are of type Split Complex.
4. The operation is performed faster when the output matrix and input vector are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmsub

Split Complex Matrix Subtract

Function Definitions

```
void zmsub(
                    float *cmxr,
                    float *cmxi,
                    long   xrows,
                    long   xcols,
                    float *cmyr,
                    float *cmyi,
                    long   yrows,
                    long   ycols,
                    float *cmzr,
                    float *cmzi);

void zmsub_(
                    float *cmxr,
                    float *cmxi,
                    long  *xrows,
                    long  *xcols,
                    float *cmyr,
                    float *cmyi,
                    long  *yrows,
                    long  *ycols,
                    float *cmzr,
                    float *cmzi);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
cmyr	input	Split complex input matrix, real
cmyi	input	Split complex input matrix, imaginary components
yrows	input	Number of rows in matrix y
ycols	input	Number of columns in matrix y
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components

Return Value

None: void function.

Description

This function subtracts the complex split matrix $(\mathbf{yr}, \mathbf{yi})$ from the complex split matrix $(\mathbf{xr}, \mathbf{xi})$ giving the complex split matrix $(\mathbf{zr}, \mathbf{zi})$. The resulting matrix has dimensions **xrows** by **xcols**.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All matrices are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmtranspose

Split Complex Matrix Transpose

Function Definitions

```
void zmtranspose(
                    float         *cmxr,
                    float         *cmxi,
                    unsigned long  xrows,
                    unsigned long  xcols,
                    unsigned long  xstride,
                    float         *cmzr,
                    float         *cmzi,
                    unsigned long  zstride);

void zmtranspose_(
                    float         *cmxr,
                    float         *cmxi,
                    unsigned long *xrows,
                    unsigned long *xcols,
                    unsigned long *xstride,
                    float         *cmzr,
                    float         *cmzi,
                    unsigned long *zstride);
```

Parameters

Argument	I/O	Description
cmxr	input	Split complex input matrix, real components
cmxi	input	Split complex input matrix, imaginary components
xrows	input	Number of rows in matrix x
xcols	input	Number of columns in matrix x
xstride	input	Row stride of the input matrix
cmzr	output	Split complex output matrix, real
cmzi	output	Split complex output matrix, imaginary components
zstride	input	Row stride of the output matrix

Return Value

None: void function.

Description

This function transposes an input matrix or sub-matrix. The input matrix has dimensions nr by nc. The input row stride is the distance, in samples, from one row to the next. Normally, rstride is the same as nc. However, a sub-matrix can be transposed by specifying the actual number of columns in the input vector as rstride and the number of columns in the submatrix as nc. Note that nc is always $<=$ rstride. The resulting matrix is deposited at mout, and has dimensions nc by nr. Note that the output stride is normally equal to nr. However, if the result is a submatrix of a larger matrix, then orstride is the number of columns in the output matrix, and nr is the number of columns in the transposed submatrix. If the matrix is square, the function can be performed in-place. Otherwise, min must not equal mout.

Restrictions and Notes

1. The operation is performed faster when the matrix row and column lengths are a power of two.
2. All rows in each matrix are of type compact.
3. All matrices are of type Split Complex.
4. The operation is performed faster when the matrices are fully aligned for the target hardware.
5. This operation is only supported in single precision.

zmult

Split Complex Multiply

Function Definitions

```
void zmult(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zmult_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
yinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function multiplies the complex split vector (xr, xi) by the complex deinterleave vector (yr, yi) giving the complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zmultconj

Split Complex Multiply Conjugate

Function Definitions

```
void zmultconj(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zmultconj_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function multiplies the complex split vector (xr, xi) by the complex conjugate of the de-interleave vector (yr, yi) giving the complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zmults

Split Complex Multiply Scalar

Function Definitions

```
void zmults(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float  csyr,
                    float  csyi,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zmults_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *csyr,
                    float *csyi,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvzinc	input	Input vector stride
cvyr	input	Split complex input scalar, real
cvyi	input	Split complex input scalar, imaginary
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function multiplies the complex split vector (xr, xi) by the complex scalar (yr, yi) giving the complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zneg

Split Complex Negate

Function Definitions

```
void zneg(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zneg_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function copies the complex split vector (xr, xi) to complex split vector (zr, zi), negating both the real and imaginary components.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zphase

Phase of Split Complex Vector

Function Definitions

```
void zphase(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *rvz,
                    long   rvzinc,
                    long   n);

void zphase_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *rvz,
                    long  *rvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
rvz	output	Output vector, real
rvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the phase angle for each complex input sample. The resulting real vector is placed at yr. The phase angle of a complex value $x + iy$ is $atan(y/x)$, adjusted so that the resulting angle is in the proper quadrant of the complex plane.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zrecip

Split Complex Reciprocal

Function Definitions

```
void zrecip(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zrecip_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	output	Split complex output vector, real
cvyi	output	Split complex output vector, imaginary components
cvyinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes the reciprocal of all elements in the complex split vector (xr, xi). The resulting complex vector is placed at (yr, yi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zsub

Split Complex Subtract

Function Definitions

```
void zsub(
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zsub_(
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvxr	input	Split complex Input vector, real components
cvxi	input	Split complex Input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex Input vector, real
cvyi	input	Split complex Input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function subtracts the complex split vector (xr, xi) from the complex deinterleave vector (yr, yi) giving the complex split vector (zr, zi).

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zvma

Split Complex Multiply/Add

Function Definitions

```
void zvma(
                    float *cvwr,
                    float *cvwi,
                    long   cvwinc,
                    float *cvxr,
                    float *cvxi,
                    long   cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zvma_(
                    float *cvwr,
                    float *cvwi,
                    long  *cvwinc,
                    float *cvxr,
                    float *cvxi,
                    long  *cvxinc,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvwr	input	Split complex input vector, real components
cvwi	input	Split complex input vector, imaginary components
cvwinc	input	Input vector stride
cvxr	input	Split complex input vector, real components
cvxi	input	Split complex input vector, imaginary components
cvxinc	input	Input vector stride
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

his function computes $z = w * x + y$ for complex split vectors z, w, x, and y

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.

2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.
4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

zvsma

Split Complex Scalar Multiply/Add

Function Definitions

```
void zvsma(
                    float *cvwr,
                    float *cvwi,
                    long   cvwinc,
                    float *csxr,
                    float *csxi,
                    float *cvyr,
                    float *cvyi,
                    long   cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long   cvzinc,
                    long   n);

void zvsma_(
                    float *cvwr,
                    float *cvwi,
                    long  *cvwinc,
                    float *csxr,
                    float *csxi,
                    float *cvyr,
                    float *cvyi,
                    long  *cvyinc,
                    float *cvzr,
                    float *cvzi,
                    long  *cvzinc,
                    long  *n);
```

Parameters

Argument	I/O	Description
cvwr	input	Split complex input vector, real components
cvwi	input	Split complex input vector, imaginary components
cvwinc	input	Input vector stride
csxr	input	Split complex input scalar pointer, real components
csxi	input	Split complex input scalar pointer, imaginary components
cvyr	input	Split complex input vector, real
cvyi	input	Split complex input vector, imaginary components
cvyinc	input	Input vector stride
cvzr	output	Split complex output vector, real
cvzi	output	Split complex output vector, imaginary components
cvzinc	input	Output vector stride
n	input	Number of samples to process

Return Value

None: void function.

Description

This function computes $\mathbf{z} = \mathbf{w} * \mathbf{s} + \mathbf{y}$ for complex split vectors **z**, **w**, **y** and scalars.

Restrictions and Notes

1. The strided elements of the input and output vectors must be equal to or greater than **n** in length.
2. The operation is performed faster when **n** is a power of two.
3. The operation is performed faster when the input and output vectors are fully aligned for the target hardware.

4. This operation is only supported in single precision.
5. All complex vectors are in Complex Split format.

5 Using Routines in the Vector Library

Header Files

Table 3.1 summarizes the six header files with which the Vector library C function prototypes are defined. Specify one or more of these header files in include statements in the first part of your C source code as required for the type of routines you are using and the manner in which your complex numbers are declared.

Filename	Function Precision	Interleaved Complex Number Arrays Declared As...
CV.h	single precision	float* array
CVC.h	single precision	float array[][2]
CVSC.h	single precision	CV_COMPLEX* array
CVD.h	double precision	double* array
CVDC.h	double precision	double array[][2]
CVDSC.h	double precision	CV_DOUBLE_COMPLEX* array

C Source Code Examples

This section includes C source code examples that show how Vector routines are called. The examples are made up of the following:

- FFT.c: Example of Complex Numbers Using Dynamically Allocated Pointers.
- FFTC.c: Example of Complex Numbers Using Two-dimensional Arrays.
- FFTSC.c: Example of Complex Numbers Using Dynamically Allocated Structures.
- realStride.c: Example Using stride With Real Data.

FFT.c, Example of Complex Numbers sing Dynamically Allocated Pointers

```
/* FFT.c
This example illustrates how arrays of complex numbers can be declared using dynamically
allocated float* or double*, and how elements of those arrays can be selectively accessed
using strides.
When this style is used, you should include the header files CV.h and/or CVD.h.
*/
#include "CV.h"
#include "CVD.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#define PI 3.141592654
extern int AV_present;
void
FFT ()
{
  /* Three dynamically-allocated arrays of single-precision complex */
  float *InputS, *OutputS, *WeightsS;
#ifdef DOUBLE
  /* Three dynamically-allocated arrays of double-precision complex */
  double *InputD, *OutputD, *WeightsD;
#endif
/* Two statically declared single-precision complex scalars */
  float DotProductS[2];
  float RotationS[2] = { 0.0, 1.0 };    /* Cosine and Sine of PI/2 */
#ifdef DOUBLE
/* Two statically declared double-precison complex scalars */
  double DotProductD[2];
  double RotationD[2] = { 0.0, 1.0 };   /* Cosine and Sine of PI/2 */
  long ExitType = EXIT_FAILURE;
#endif
  long i, fc, NumSamples = 8;
  AV_present = 1;
/* Dynamically allocate memory for the Weights, Input and Output tables */
  WeightsS = (float *) memalign (32, NumSamples * 2 * sizeof (float));
  if (WeightsS == 0)
    {
      printf ("Error Allocating WeightsS\n");
      goto exit6;
    }
#ifdef DOUBLE
  WeightsD = (double *) memalign (32, NumSamples * 2 * sizeof (double));
  if (WeightsD == 0)
    {
      printf ("Error Allocating WeightsD\n");
      goto exit5;
    }
#endif
  InputS = (float *) memalign (32, NumSamples * 2 * sizeof (float));
  if (InputS == 0)
    {
      printf ("Error Allocating InputS\n");
      goto exit4;
```

```
    }
#ifdef DOUBLE
  InputD = (double *) memalign (32, NumSamples * 2 * sizeof (double));
  if (InputD == 0)
    {
      printf ("Error Allocating InputD\n");
      goto exit3;
    }
#endif
  OutputS = (float *) memalign (32, NumSamples * 2 * sizeof (float));
  if (OutputS == 0)
    {
      printf ("Error Allocating OutputS\n");
      goto exit2;
    }
#ifdef DOUBLE
  OutputD = (double *) memalign (32, NumSamples * 2 * sizeof (double));
  if (OutputD == 0)
    {
      printf ("Error Allocating OutputD\n");
      goto exit1;
    }
#endif
  /* Initialize the Weights tables */
  fftwts (WeightsS, NumSamples, NumSamples);
#ifdef DOUBLE
  fftwtsd (WeightsD, NumSamples, NumSamples);
#endif
  /*
     Initialize the single-precision Input table with data that gives us systematic results
   */
  for (i = 0; i < 2 * NumSamples; i += 2)
    {
      InputS[i] = 0.0;
      InputS[i + 1] = 0.0;
    }
  for (fc = 1; fc <= NumSamples; fc++)
    for (i = 0; i < 2 * NumSamples; i += 2)
      {
        InputS[i] += (float) fc *(float) cos (fc * (i * PI / NumSamples));
        InputS[i + 1] += (float) fc *(float) sin (fc * (i * PI / NumSamples));
      }
  printf ("\n\nSingle-Precision Input Data (time domain)\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
    printf ("%2ld%12.4f%12.4f\n", i / 2, InputS[i], InputS[i + 1]);
  /* Copy the single-precision Input data into the double-precision Input table */
#ifdef DOUBLE
  vspdp (InputS, 1, InputD, 1, 2 * NumSamples);
  printf ("\n\nDouble-Precision Input Data (time domain)\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
    printf ("%2ld%12.4f%12.4f\n", i / 2, InputD[i], InputD[i + 1]);
#endif
  /* Perform a single-precision FFT, scale and print the results */
  cfftf (InputS, 2, OutputS, 2, NumSamples);
  cfftsc (OutputS, NumSamples);
  printf ("\n\n\nSingle Precision Output Data (frequency domain)\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
```

```
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputS[i], OutputS[i + 1]);
  /* Rotate the single-precision results through PI/2 and print them out again */
  cvcsml (OutputS, 2, RotationS, OutputS, 2, NumSamples);
  printf ("\n\n\nSingle Precision Output Data after rotation\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputS[i], OutputS[i + 1]);
  /* Compute the dot product of OutputS times itself and print out the results */
  cdotpr (OutputS, 2, OutputS, 2, DotProductS, NumSamples);
  printf
    ("\n\nSingle Precision Dot Product of all %ld Points= %12.4f%12.4f\n",
     NumSamples, DotProductS[0], DotProductS[1]);
  /* Using "stride," compute the dot product of odd numbered elements of OutputS
     and print out the results
   */
  cdotpr (&OutputS[2], 4, &OutputS[2], 4, DotProductS, NumSamples / 2);
  printf ("\n\nSingle Precision Dot Product of the %ld Odd Numbered Points=
%12.4f%12.4f\n", NumSamples / 2, DotProductS[0], DotProductS[1]);
#ifdef DOUBLE
  /* Perform a double-precision FFT, scale and print the results */
  cfftfd (InputD, 2, OutputD, 2, NumSamples);
  cfftscd (OutputD, NumSamples);
  printf ("\n\n\nDouble-Precision Output Data (frequency domain)\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputD[i], OutputD[i + 1]);
  /* Rotate the double-precision results through PI/2 and print them out again */
  cvcsmld (OutputD, 2, RotationD, OutputD, 2, NumSamples);
  printf ("\n\n\nDouble Precision Output Data after rotation\n\n");
  for (i = 0; i < 2 * NumSamples; i += 2)
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputD[i], OutputD[i + 1]);
  /* Compute the dot product of OutputD times itself and print out the results */
  cdotprd (OutputD, 2, OutputD, 2, DotProductD, NumSamples);
  printf ("\n\nDouble Precision Dot Product = %12.4f%12.4f\n", DotProductD[0],
          DotProductD[1]);
  /* Using "stride," compute the dot product of odd numbered elements of OutputD
     and print out the results
   */
  cdotprd (&OutputD[2], 4, &OutputD[2], 4, DotProductD, NumSamples / 2);
  printf ("\n\nDouble Precision Dot Product of the %ld Odd Numbered Points=
%12.4f%12.4f\n", NumSamples / 2, DotProductD[0], DotProductD[1]);
  /* Release the dynamically allocated memory and exit */
  ExitType = EXIT_SUCCESS;
  free (OutputD);
exit1:
#endif
  free (OutputS);
exit2:
#ifdef DOUBLE
  free (InputD);
exit3:
#endif
  free (InputS);
exit4:
#ifdef DOUBLE
  nasCW_freewtsd (WeightsD);
exit5:
#endif
  nasCW_freewts (WeightsS);
```

```
exit6:
  return;
}

int
main (void)
{
  nas_init();
  FFT ();
  nas_finalize();
  return 0;
}
```

FFTC.c, Example of Complex Numbers Using Two-dimensional Arrays

```
/* FFTC.c
This example illustrates how arrays of complex numbers can be declared as statically
allocated two-dimensional arrays of floats or doubles, and how elements of those arrays can
be selectively accessed using strides.
When this style is used, you should include the header files CVC.h and/or CVDC.h.
*/
#include "CVC.h"
#include "CVDC.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define PI 3.141592654
int
FFTC ()
{
  /* Three statically-allocated arrays of single-precision complex */
  float InputS[8][2], OutputS[8][2], WeightsS[8][2];
#ifdef DOUBLE
  /* Three statically-allocated arrays of double-precision complex */
  double InputD[8][2], OutputD[8][2], WeightsD[8][2];
#endif
  /* Two statically declared single-precision complex scalars */
  float DotProductS[2];
  float RotationS[2] = { 0.0, 1.0 };    /* Cosine and Sine of PI/2 */
#ifdef DOUBLE
  /* Two statically declared double-precison complex scalars */
  double DotProductD[2];
  double RotationD[2] = { 0.0, 1.0 };   /* Cosine and Sine of PI/2 */
#endif
  long i, fc, NumSamples = 8;
  /* Initialize the weights tables */
  fftwts (WeightsS, NumSamples, NumSamples);
#ifdef DOUBLE
  fftwtsd (WeightsD, NumSamples, NumSamples);
#endif
  /* Initialize the single-precision Input table with data that gives us systematic
     results */
  for (i = 0; i < NumSamples; i++)
    {
      InputS[i][0] = 0.0;
      InputS[i][1] = 0.0;
    }
  for (fc = 1; fc <= NumSamples; fc++)
```

```
      for (i = 0; i < NumSamples; i++)
        {
          InputS[i][0] +=
            (float) fc *(float) cos (fc * (i * 2 * PI / NumSamples));
          InputS[i][1] +=
            (float) fc *(float) sin (fc * (i * 2 * PI / NumSamples));
        }
    printf ("\n\nSingle-Precision Input Data (time domain)\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, InputS[i][0], InputS[i][1]);
  #ifdef DOUBLE
    /* Copy the single-precision Input data into the double-precision Input table */
    vspdp ((float *) InputS, 1, (double *) InputD, 1, 2 * NumSamples);
    printf ("\n\nDouble-Precision Input Data (time domain)\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, InputD[i][0], InputD[i][1]);
  #endif
    /* Perform a single-precision FFT, scale and print the results */
    cfftf (InputS, 2, OutputS, 2, NumSamples);
    cfftsc (OutputS, NumSamples);
    printf ("\n\n\nSingle-Precision Output Data (frequency domain)\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, OutputS[i][0], OutputS[i][1]);
    /* Rotate the single-precision results through PI/2 and print them out again */
    cvcsml (OutputS, 2, RotationS, OutputS, 2, NumSamples);
    printf ("\n\n\nSingle Precision Output Data after rotation\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, OutputS[i][0], OutputS[i][1]);
    /* Compute the dot product of OutputS times itself and print out the results */
    cdotpr (OutputS, 2, OutputS, 2, DotProductS, NumSamples);
    printf ("\n\nSingle Precision Dot Product = %12.4f%12.4f\n", DotProductS[0],
            DotProductS[1]);
    /* Using "stride," compute the dot product of odd numbered elements of OutputS
       and print out the results \
     */
    cdotpr (&OutputS[1], 4, &OutputS[1], 4, DotProductS, NumSamples / 2);
    printf ("\n\nSingle Precision Dot Product of the %ld Odd Numbered Points=
  %12.4f%12.4f\n", NumSamples / 2, DotProductS[0], DotProductS[1]);
  #ifdef DOUBLE
    /* Perform a double-precision FFT, scale and print the results */
    cfftfd (InputD, 2, OutputD, 2, NumSamples);
    cfftscd (OutputD, NumSamples);
    printf ("\n\n\nDouble-Precision Output Data (frequency domain)\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, OutputD[i][0], OutputD[i][1]);
    /* Rotate the double-precision results through PI/2 and print them out again */
    cvcsmld (OutputD, 2, RotationD, OutputD, 2, NumSamples);
    printf ("\n\n\nDouble Precision Output Data after rotation\n\n");
    for (i = 0; i < NumSamples; i++)
      printf ("%2ld%12.4f%12.4f\n", i, OutputD[i][0], OutputD[i][1]);
    /* Compute the dot product of OutputD times itself and print out the results */
    cdotprd (OutputD, 2, OutputD, 2, DotProductD, NumSamples);
    printf ("\n\nDouble Precision Dot Product = %12.4f%12.4f\n", DotProductD[0],
            DotProductD[1]);
    /* Using "stride," compute the dot product of odd numbered elements of OutputD
       and print out the results
     */
```

```
  cdotprd (&OutputD[1], 4, &OutputD[1], 4, DotProductD, NumSamples / 2);
  printf ("\n\nDouble Precision Dot Product of the %ld Odd Numbered Points=
%12.4f%12.4f\n", NumSamples / 2, DotProductD[0], DotProductD[1]);
#endif
  return EXIT_SUCCESS;
}

int
main (void)
{
  nas_init();
  FFTC ();
  nas_finalize();
  return 0;
}
```

FFTSC.c, Example of Complex Numbers Using Dynamically Allocated Structures

```
/* FFTSC.c
This example illustrates how arrays of complex numbers can be declared using dynamically
allocated CV_COMPLEX* or CV_DOUBLE_COMPLEX*, and how elements of those arrays can
be selectively accessed using strides.
When this style is used, you should include the header files CVSC.h and/or CVDSC.h. In
addition to providing the function prototypes, these header files also define the structures
for the COMPLEX data types.
*/
#include "CVSC.h"
#include "CVDSC.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <malloc.h>
#define PI 3.141592654
int
FFTSC ()
{
  /* Three dynamically-allocated arrays of single-precision complex */
  CV_COMPLEX *InputS, *OutputS, *WeightsS;
#ifdef DOUBLE
  /* Three dynamically-allocated arrays of double-precision complex */
  CV_DOUBLE_COMPLEX *InputD, *OutputD, *WeightsD;
#endif
  /* Two statically declared single-precision complex scalars */
  CV_COMPLEX DotProductS;
  CV_COMPLEX RotationS = { 0.0, 1.0 };  /* Cosine and Sine of PI/2 */
#ifdef DOUBLE
  /* Two statically declared double-precison complex scalars */
  CV_DOUBLE_COMPLEX DotProductD;
  CV_DOUBLE_COMPLEX RotationD = { 0.0, 1.0 };   /* Cosine and Sine of PI/2 */
#endif
  long i, fc, NumSamples = 8, ExitType = EXIT_FAILURE;
  /* Dynamically allocate memory for the Weights, Input and Output tables */
  WeightsS = (CV_COMPLEX *) memalign (32, NumSamples * sizeof (CV_COMPLEX));
  if (WeightsS == 0)
    {
      printf ("Error Allocating WeightsS\n");
      goto exit6;
    }
```

```
#ifdef DOUBLE
  WeightsD = (CV_DOUBLE_COMPLEX *) memalign (32, NumSamples *
                                              sizeof (CV_DOUBLE_COMPLEX));
  if (WeightsD == 0)
    {
      printf ("Error Allocating WeightsD\n");
      goto exit5;
    }
#endif
  InputS = (CV_COMPLEX *) memalign (32, NumSamples * sizeof (CV_COMPLEX));
  if (InputS == 0)
    {
      printf ("Error Allocating InputS\n");
      goto exit4;
    }
#ifdef DOUBLE
  InputD = (CV_DOUBLE_COMPLEX *) memalign (32, NumSamples *
                                            sizeof (CV_DOUBLE_COMPLEX));
  if (InputD == 0)
    {
      printf ("Error Allocating InputD\n");
      goto exit3;
    }
#endif
  OutputS = (CV_COMPLEX *) memalign (32, NumSamples * sizeof (CV_COMPLEX));
  if (OutputS == 0)
    {
      printf ("Error Allocating OutputS\n");
      goto exit2;
    }
#ifdef DOUBLE
  OutputD = (CV_DOUBLE_COMPLEX *) memalign (32, NumSamples *
                                             sizeof (CV_DOUBLE_COMPLEX));
  if (OutputD == 0)
    {
      printf ("Error Allocating OutputD\n");
      goto exit1;
    }
#endif
  /* Initialize the Weights tables */
  fftwts (WeightsS, NumSamples, NumSamples);
#ifdef DOUBLE
  fftwtsd (WeightsD, NumSamples, NumSamples);
#endif
  /* Initialize the single-precision Input table with data that gives us systematic
     results */
  for (i = 0; i < NumSamples; i++)
    {
      InputS[i].r = 0.0;
      InputS[i].i = 0.0;
    }
  for (fc = 1; fc <= NumSamples; fc++)
    for (i = 0; i < NumSamples; i++)
      {
        InputS[i].r +=
          (float) fc *(float) cos (fc * (i * 2 * PI / NumSamples));
        InputS[i].i +=
```

```
            (float) fc *(float) sin (fc * (i * 2 * PI / NumSamples));
        }
  printf ("\n\nSingle-Precision Input Data (time domain)\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i, InputS[i].r, InputS[i].i);
#ifdef DOUBLE
  /* Copy the single-precision Input data into the double-precision Input table */
  vspdp ((float *) InputS, 1, (double *) InputD, 1, 2 * NumSamples);
  printf ("\n\nDouble-Precision Input Data (time domain)\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i, InputD[i].r, InputD[i].i);
#endif
  /* Perform a single-precision FFT, scale and print the results */
  cfftf (InputS, 2, OutputS, 2, NumSamples);
  cfftsc (OutputS, NumSamples);
  printf ("\n\n\nSingle-Precision Output Data (frequency domain)\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i, OutputS[i].r, OutputS[i].i);
  /* Rotate the single-precision results through PI/2 and print them out again */
  cvcsml (OutputS, 2, &RotationS, OutputS, 2, NumSamples);
  printf ("\n\n\nSingle Precision Output Data after rotation\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputS[i].r, OutputS[i].i);
  /* Compute the dot product of OutputS times itself and print out the results */
  cdotpr (OutputS, 2, OutputS, 2, &DotProductS, NumSamples);
  printf ("\n\nSingle Precision Dot Product = %12.4f%12.4f\n", DotProductS.r,
          DotProductS.i);
  /* Using "stride," compute the dot product of odd numbered elements of OutputS
     and print out the results
   */
  cdotpr (&OutputS[1], 4, &OutputS[1], 4, &DotProductS, NumSamples / 2);
  printf ("\n\nSingle Precision Dot Product of the %ld Odd Numbered Points=
%12.4f%12.4f\n", NumSamples / 2, DotProductS.r, DotProductS.i);
#ifdef DOUBLE
  /* Perform a double-precision FFT, scale and print the results */
  cfftfd (InputD, 2, OutputD, 2, NumSamples);
  cfftscd (OutputD, NumSamples);
  printf ("\n\n\nDouble-Precision Output Data (frequency domain)\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i, OutputD[i].r, OutputD[i].i);
  /* Rotate the double-precision results through PI/2 and print them out again */
  cvcsmld (OutputD, 2, &RotationD, OutputD, 2, NumSamples);
  printf ("\n\n\nDouble Precision Output Data after rotation\n\n");
  for (i = 0; i < NumSamples; i++)
    printf ("%2ld%12.4f%12.4f\n", i / 2, OutputD[i].r, OutputD[i].i);
  /* Compute the dot product of OutputD times itself and print out the results */
  cdotprd (OutputD, 2, OutputD, 2, &DotProductD, NumSamples);
  printf ("\n\nDouble Precision Dot Product = %12.4f%12.4f\n", DotProductD.r,
          DotProductD.i);
  /* Using "stride," compute the dot product of odd numbered elements of OutputD
     and print out the results
   */
  cdotprd (&OutputD[1], 4, &OutputD[1], 4, &DotProductD, NumSamples / 2);
  printf ("\n\nDouble Precision Dot Product of the %ld Odd Numbered Points=
%12.4f%12.4f\n", NumSamples / 2, DotProductS.r, DotProductS.i);
  /* Release the dynamically allocated memory and exit */
  ExitType = EXIT_SUCCESS;
```

```
  free (OutputD);
exit1:
#endif
  free (OutputS);
exit2:
#ifdef DOUBLE
  free (InputD);
exit3:
#endif
  free (InputS);
exit4:
#ifdef DOUBLE
  nasCW_freewtsd (WeightsD);
exit5:
#endif
  nasCW_freewts (WeightsS);
exit6:
  return ExitType;
}

int
main (void)
{
  nas_init();
  FFTSC ();
  nas_finalize();
  return 0;
}
```

realStride.c, Example Using stride With Real Data

```
/*
Here are some simple examples of using strides on real input vector data.
 *
This only uses the vsma function to add two input vectors but in general increments for all
real vectors are the same.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include "CV.h"
/* define one compact real input vector of 10 elements */
float in1[] = { 0., 1., 2., 3., 4., 5., 6., 7., 8., 9. };

/* define a second compact real input vector of 10 elements */
float in2[] = { 10., 20., 30., 40., 50., 60., 70., 80., 90., 100. };

float out[10];
/* define the constant multiplier */
float rs1 = 1.0;
/*
 *************** printVecs *****************
 *
 * Print the input and output data
 *
 *******************************************
 */
void
```

```
printVecs ()                    /* print input and output vectors */
{
  int i;
  printf (" i In1 + In2 = out\n");
  for (i = 0; i < 10; i++)
    {
      printf ("%3d %3.2f, %3.2f, %3.2f\n", i, in1[i], in2[i], out[i]);
    }
}

/*
 **************** clearOut *****************
 *
 * Set the output vector to zero
 *
 *******************************************
 */
void
clearOut ()
{
  int i;
  for (i = 0; i < 10; i++)
    {
      out[i] = 0;
    }
}
/*
 **************** main *********************
 *
 * Do some examples of strided real vectors
 *
 *******************************************
 */
void
realStride ()
{
  printf ("*****************************************\n");
  printf ("*\n");
  printf ("* First just add each element of the input 1\n");
  printf ("* vector to the corresponding input of the \n");
  printf ("* other input vector\n");
  printf ("* e.g. increment of each vector is 1\n");
  printf ("*\n");
  printf ("*****************************************\n\n");
  vsma (in1, 1, &rs1, in2, 1, out, 1, 10);
  printVecs ();
  printf ("*****************************************\n");
  printf ("*\n");
  printf ("* Now clear the output, and add the adjacent\n");
  printf ("* elements of input 1 to every other element \n");
  printf ("* of input 2 over 5 elements\n");
  printf ("* e.g. increment one is 1 increment two is 2\n");
  printf ("* output increment is 1\n");
  printf ("*\n");
  printf ("*****************************************\n\n");
  clearOut ();
  vsma (in1, 1, &rs1, in2, 2, out, 1, 5);
```

```
  printVecs ();
  printf ("*****************************************\n");
  printf ("*\n");
  printf ("* Now clear the output, and add every other\n");
  printf ("* element of input 1 to adjacent elements \n");
  printf ("* of input 2 over 5 elements\n");
  printf ("* e.g. increment one is 2 increment two is 1\n");
  printf ("* output increment is 1\n");
  printf ("*\n");
  printf ("*****************************************\n\n");
  clearOut ();
  vsma (in1, 2, &rs1, in2, 1, out, 1, 5);
  printVecs ();
  printf ("*****************************************\n");
  printf ("*\n");
  printf ("* Now clear the output, and add every other\n");
  printf ("* element of input 1 to adjacent elements \n");
  printf ("* of input 2 over 5 elements, storing result\n");
  printf ("* in every other output element\n");
  printf ("* e.g. increment one is 2 increment two is 1\n");
  printf ("* output increment is 2\n");
  printf ("*\n");
  printf ("*****************************************\n\n");
  clearOut ();
  vsma (in1, 2, &rs1, in2, 1, out, 2, 5);
  printVecs ();
}

int
main (void)
{
  nas_init();
  realStride ();
  nas_finalize();
  return 0;
}
```

Makefile

After writing your C source code incorporating whatever functions are needed from the subroutine library, create a makefile with which to create an executable. The makefile must define the proper paths for the library itself (CV.a), typically located in /usr/lib, and the header files, typically located in /usr/include. Makefiles require a specific format as determined by the operating system used. Refer to the realStride.c makefile examples for Linux below. Running the makefile creates an object code file which is linked to create the executable program. The linking is usually done right after the compiling during the course of the makefile session.

Example of Makefile for Linux

```
SRC = realStride.c
OBJ = realStride.o
CVLIBDIR = /usr/lib
CVLIBINC = /usr/include
CVLIB = CV_lx
CCOPT = -lm
PROGS = realStride
CC = gcc
sync: $(OBJ)
```

```
$(CC) $(OBJ) -L$(CVLIBDIR) -l$(CVLIB) $(CCOPT) -o $(PROGS)
$(OBJ):$(SRC)
$(CC) -I$(CVLIBINC) -c $(SRC)
clean:
rm -f *.o
exe:
```

6 GLOSSARY

Admitted	*Block* state where the *data array* (memory) and associated *views* are available for VSIPL computations, and not available for user I/O or access.
Attribute	Characteristic or state of an object, such as *admitted* / *released*, *stride*, or *length*.
Binary Function	A function with two input arguments.
Block	A data storage abstraction representing contiguous data elements consisting of a *data array* and a VSIPL *block object*.
Block Object	Descriptor for a *data array* and its *attributes*, including a reference to the data array, the state of the block, data type and size.
Block Offset	The number of *elements* from the start of a *block*. A view with a block offset of zero starts at the beginning of the block.
Boolean	Used to represent the values of true and false, where false is always zero, and true is non-zero.
Bound	A *view* or *block* is bound to a *data array* if it references the data array.
Cloned View	An exact duplicate of a *view object*.
Column	Rightmost dimension in a *matrix*.
Column Stride	The number of *block elements* between successive elements within a *column*.
Complex Block	*Block* containing only complex *elements*. There are two formats for released complex blocks – *split* and *interleaved*. The complex data format for admitted complex blocks is not known to the user.
Conformant Views	*Views* that are the correct shape/size for a given computation.
const Object	An object that is not modified by the function, although data referenced by the const object may be modified.
Create	To allocate memory for an object and initialise it (if appropriate).
Data Array	Memory where data is stored.
Derived Block	A *real block* derived from a *complex block*. Note that the only way to create a derived block is to create a *derived view* of the real or complex component of a *split* complex view. In all other cases, retrieving the block from a view returns a reference to the original block.

Derived View	A derived view is a *view* created using a VSIPL function whose arguments include another view (a parent view). The derived view's data is some subset of the parent view's data. The data subset depends on the function call, and is physically co-located in memory with the parent view's data.
Destroy	To release the memory allocated to an object.
Development Library	An implementation of VSIPL that maximises error reporting at the possible expense of performance.
Domain	The set of all valid input values to a function.
Element	The atomic portion of data associated with a *block* or a *view*. For example, an element of a *complex block* of precision double is a complex number of precision double; for a view of type float an element is a single float number.
Hermitian Transpose	Conjugate transpose.
Hint	Information provided by the user to some VSIPL functions to aid optimization. Hints are optional and may be ignored by the implementation. Wrong hints may result in incorrect behavior.
In-Place	A type of algorithm implementation in which the memory used to hold the input to an algorithm is overwritten (completely or partially) with the output data. Often referred to in the context of an FFT algorithm.
Interleaved Complex	Storage format for *user data arrays* where the real and complex *element* components alternate in physical memory.
Kernel	The filter vector used in a FIR filter, or the vector or matrix used as the weights in a convolution.
Length	Number of *elements* in a view along a *view dimension*.
Matrix	A two-dimensional view.
Opaque	An opaque object may not be manipulated by simple assignment statements. Its attributes must be set/retrieved through access functions. All VSIPL objects are opaque.
Out-of-place	If none of the output views in a function call overlap the input views, the function is considered out-of-place.
Overlapped	Indicates that two or more *views* or *blocks* share one or more memory locations.
Production Library	A VSIPL implementation that maximises performance at the possible expense of not detecting user errors.
Range	Valid output values from a function.
Real Block	A *block* containing only real *elements*.
Region of Support	For neighborhood operations (i.e. FIR filtering, convolution), the non-zero values in the kernel, or the output. For example, a 3×3 FIR filter has a 'kernel region of support' of 3×3.
Released	*Block* state where the associated *data array* is available for user I/O and application access, but not available for VSIPL computations.
Row	Left-most dimension of a *matrix*.
Row Stride	The number of *block* elements between successive elements within a *row*.
Split Complex	Storage format for released *complex blocks* where the real *element* components are stored in one physically contiguous *data array*, and the imaginary components are stored in a separate physically contiguous data array.
Stride	Distance between successive *elements* of the block data array in a *view* along a view dimension. Strides can be positive, negative, or zero.
Subview	A *derived view* that describes a subset of the data from the original view, and is the same type as the original view.
Tensor	An n-dimensional matrix. VSIPL only supports three-dimensional tensors (3-tensor). The three dimensions are referred to as X, Y and Z.
Ternary Function	A function with three input arguments.
Unary Function	A function with a single input argument.
User Block	A block which is associated with user data arrays. User blocks are created in the released state and may be admitted and released.
User Data Array	Memory that has been allocated by the application for the storage of data using some functionality not part of the VSIPL standard.
Vector	A one-dimensional *view*.

View	A portion of a *block*, and a *view object* describing it. The view object has structural information allowing the data to be interpreted as a one-, two- or three-dimensional array for arithmetic processing.
View Dimension	A *view* represents a one-, two-, or three-dimensional data organisation termed respectively a *vector*, *matrix* or *tensor*. A *view dimension* represents one of the standard directions of these data representations.
View Object	A description of a portion of a *block*, including structural information that allows the data to be interpreted as a one-, two- or three-dimensional array for arithmetic processing. Attributes of the *view object* include *offset*, *stride*(s) and *length*(s).
VSIPL Block	*Block* referencing or *bound* to VSIPL data. A VSIPL block is created in the *admitted* state and may not be *released*.
VSIPL Data Array	Memory that has been allocated for the storage of data using some functionality that is part of the VSIPL standard.