

ReproBLAS

Generated by Doxygen 1.8.11

Contents

1	File Index	1
1.1	File List	1
2	File Documentation	3
2.1	include/binned.h File Reference	3
2.1.1	Detailed Description	10
2.1.2	Macro Definition Documentation	11
2.1.2.1	binned_DBCAPACITY	11
2.1.2.2	binned_DBENDURANCE	11
2.1.2.3	binned_DBMAXFOLD	12
2.1.2.4	binned_DBMAXINDEX	12
2.1.2.5	binned_DMCOMPRESSION	12
2.1.2.6	binned_DMEXPANSION	12
2.1.2.7	binned_SBCAPACITY	13
2.1.2.8	binned_SBENDURANCE	13
2.1.2.9	binned_SBMAXFOLD	13
2.1.2.10	binned_SBMAXINDEX	14
2.1.2.11	binned_SMCOMPRESSION	14
2.1.2.12	binned_SMEXPANSION	14
2.1.2.13	DBWIDTH	15
2.1.2.14	SBWIDTH	15
2.1.3	Typedef Documentation	15
2.1.3.1	double_binned	15
2.1.3.2	double_complex_binned	16

2.1.3.3	<code>float_binned</code>	16
2.1.3.4	<code>float_complex_binned</code>	16
2.1.4	Function Documentation	16
2.1.4.1	<code>binned_cballocc(const int fold)</code>	16
2.1.4.2	<code>binned_cbcadd(const int fold, const void *X, float_complex_binned *Y)</code>	17
2.1.4.3	<code>binned_cbcadd(const int fold, const float_complex_binned *X, float_complex↔ _binned *Y)</code>	17
2.1.4.4	<code>binned_cbcaddv(const int fold, const int N, const float_complex_binned *↔ X, const int incX, float_complex_binned *Y, const int incY)</code>	18
2.1.4.5	<code>binned_cbcset(const int fold, const float_complex_binned *X, float_complex↔ binned *Y)</code>	18
2.1.4.6	<code>binned_cbcconv(const int fold, const void *X, float_complex_binned *Y)</code>	19
2.1.4.7	<code>binned_cbcdeposit(const int fold, const void *X, float_complex_binned *Y)</code>	19
2.1.4.8	<code>binned_cbcupdate(const int fold, const void *X, float_complex_binned *Y)</code>	20
2.1.4.9	<code>binned_cbnegate(const int fold, float_complex_binned *X)</code>	20
2.1.4.10	<code>binned_cbnum(const int fold)</code>	20
2.1.4.11	<code>binned_cbprint(const int fold, const float_complex_binned *X)</code>	21
2.1.4.12	<code>binned_cbreorm(const int fold, float_complex_binned *X)</code>	21
2.1.4.13	<code>binned_cbsbset(const int fold, const float_binned *X, float_complex_binned *Y)</code>	22
2.1.4.14	<code>binned_cbsetzero(const int fold, float_complex_binned *X)</code>	22
2.1.4.15	<code>binned_cbsize(const int fold)</code>	23
2.1.4.16	<code>binned_cbsupdate(const int fold, const float X, float_complex_binned *Y)</code>	23
2.1.4.17	<code>binned_ccbconv_sub(const int fold, const float_complex_binned *X, void *conv)</code>	23
2.1.4.18	<code>binned_ccmconv_sub(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, void *conv)</code>	24
2.1.4.19	<code>binned_cmcadd(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	24
2.1.4.20	<code>binned_cmconv(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	25
2.1.4.21	<code>binned_cmcdeposit(const int fold, const void *X, float *priY, const int incpriY)</code>	25
2.1.4.22	<code>binned_cmcadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	26
2.1.4.23	<code>binned_cmcset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	26

2.1.4.24	<code>binned_cmcupdate(const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	27
2.1.4.25	<code>binned_cmddenorm(const int fold, const float *priX)</code>	27
2.1.4.26	<code>binned_cmnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	28
2.1.4.27	<code>binned_cmprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	28
2.1.4.28	<code>binned_cmrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	29
2.1.4.29	<code>binned_cmsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	29
2.1.4.30	<code>binned_cmsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	30
2.1.4.31	<code>binned_cmsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	31
2.1.4.32	<code>binned_cmsupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	32
2.1.4.33	<code>binned_dballoc(const int fold)</code>	32
2.1.4.34	<code>binned_dbbound(const int fold, const int N, const double X, const double S)</code>	33
2.1.4.35	<code>binned_dbdadd(const int fold, const double X, double_binned *Y)</code>	33
2.1.4.36	<code>binned_dbdbadd(const int fold, const double_binned *X, double_binned *Y)</code>	34
2.1.4.37	<code>binned_dbdbaddsq(const int fold, const double scaleX, const double_binned *X, const double scaleY, double_binned *Y)</code>	34
2.1.4.38	<code>binned_dbdbaddv(const int fold, const int N, const double_binned *X, const int incX, double_binned *Y, const int incY)</code>	35
2.1.4.39	<code>binned_dbdbset(const int fold, const double_binned *X, double_binned *Y)</code>	35
2.1.4.40	<code>binned_dbdconv(const int fold, const double X, double_binned *Y)</code>	36
2.1.4.41	<code>binned_dbddeposit(const int fold, const double X, double_binned *Y)</code>	36
2.1.4.42	<code>binned_dbdupdate(const int fold, const double X, double_binned *Y)</code>	37
2.1.4.43	<code>binned_dbnegate(const int fold, double_binned *X)</code>	37
2.1.4.44	<code>binned_dbnum(const int fold)</code>	38
2.1.4.45	<code>binned_dbprint(const int fold, const double_binned *X)</code>	38
2.1.4.46	<code>binned_dbrenorm(const int fold, double_binned *X)</code>	38
2.1.4.47	<code>binned_dbsetzero(const int fold, double_binned *X)</code>	39
2.1.4.48	<code>binned_dbsize(const int fold)</code>	39

2.1.4.49	<code>binmed_dbdconv(const int fold, const double_binned *X)</code>	40
2.1.4.50	<code>binmed_ddmconv(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	40
2.1.4.51	<code>binmed_dindex(const double X)</code>	41
2.1.4.52	<code>binmed_dmbins(const int X)</code>	41
2.1.4.53	<code>binmed_dmdadd(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	42
2.1.4.54	<code>binmed_dmdconv(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	42
2.1.4.55	<code>binmed_dmddeposit(const int fold, const double X, double *priY, const int incpriY)</code>	43
2.1.4.56	<code>binmed_dmdenorm(const int fold, const double *priX)</code>	43
2.1.4.57	<code>binmed_dmdmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.58	<code>binmed_dmdmaddsq(const int fold, const double scaleX, const double *priX, const int incpriX, const double *carX, const int inccarX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	44
2.1.4.59	<code>binmed_dmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	45
2.1.4.60	<code>binmed_dmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	45
2.1.4.61	<code>binmed_dmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	46
2.1.4.62	<code>binmed_dmindex(const double *priX)</code>	46
2.1.4.63	<code>binmed_dmindex0(const double *priX)</code>	47
2.1.4.64	<code>binmed_dmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	47
2.1.4.65	<code>binmed_dmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	48
2.1.4.66	<code>binmed_dmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	48
2.1.4.67	<code>binmed_dmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	49
2.1.4.68	<code>binmed_dscale(const double X)</code>	49
2.1.4.69	<code>binmed_sballoc(const int fold)</code>	50
2.1.4.70	<code>binmed_sbbound(const int fold, const int N, const float X, const float S)</code>	50

2.1.4.71	<code>binned_sbnegate(const int fold, float_binned *X)</code>	51
2.1.4.72	<code>binned_sbnorm(const int fold)</code>	51
2.1.4.73	<code>binned_sbprint(const int fold, const float_binned *X)</code>	52
2.1.4.74	<code>binned_sbnorm(const int fold, float_binned *X)</code>	52
2.1.4.75	<code>binned_sbsadd(const int fold, const float X, float_binned *Y)</code>	53
2.1.4.76	<code>binned_sbsbadd(const int fold, const float_binned *X, float_binned *Y)</code>	53
2.1.4.77	<code>binned_sbsbadds(const int fold, const float scaleX, const float_binned *X, const float scaleY, float_binned *Y)</code>	54
2.1.4.78	<code>binned_sbsbaddv(const int fold, const int N, const float_binned *X, const int incX, float_binned *Y, const int incY)</code>	54
2.1.4.79	<code>binned_sbsbset(const int fold, const float_binned *X, float_binned *Y)</code>	55
2.1.4.80	<code>binned_sbsbze(const int fold)</code>	55
2.1.4.81	<code>binned_sbsconv(const int fold, const float X, float_binned *Y)</code>	55
2.1.4.82	<code>binned_sbsdeposit(const int fold, const float X, float_binned *Y)</code>	56
2.1.4.83	<code>binned_sbsetzero(const int fold, float_binned *X)</code>	56
2.1.4.84	<code>binned_sbsupdate(const int fold, const float X, float_binned *Y)</code>	57
2.1.4.85	<code>binned_sindex(const float X)</code>	57
2.1.4.86	<code>binned_smbins(const int X)</code>	58
2.1.4.87	<code>binned_smdenorm(const int fold, const float *priX)</code>	58
2.1.4.88	<code>binned_smindex(const float *priX)</code>	59
2.1.4.89	<code>binned_smindex0(const float *priX)</code>	59
2.1.4.90	<code>binned_smnegate(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	60
2.1.4.91	<code>binned_smprint(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	60
2.1.4.92	<code>binned_smrenorm(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	61
2.1.4.93	<code>binned_smsadd(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	61
2.1.4.94	<code>binned_smsconv(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	62
2.1.4.95	<code>binned_smsdeposit(const int fold, const float X, float *priY, const int incpriY)</code>	62
2.1.4.96	<code>binned_smsetzero(const int fold, float *priX, const int incpriX, float *carX, const int inccarX)</code>	63

2.1.4.97	<code>binned_smsmadd(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	63
2.1.4.98	<code>binned_smsmaddsq(const int fold, const float scaleX, const float *priX, const int incpriX, const float *carX, const int inccarX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	64
2.1.4.99	<code>binned_smsmset(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	64
2.1.4.100	<code>binned_smsrescale(const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	65
2.1.4.101	<code>binned_smsupdate(const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)</code>	66
2.1.4.102	<code>binned_ssbconv(const int fold, const float_binned *X)</code>	67
2.1.4.103	<code>binned_sscales(const float X)</code>	67
2.1.4.104	<code>binned_ssmconv(const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)</code>	68
2.1.4.105	<code>binned_ufp(const double X)</code>	68
2.1.4.106	<code>binned_ufpf(const float X)</code>	69
2.1.4.107	<code>binned_zballoc(const int fold)</code>	69
2.1.4.108	<code>binned_zbdbset(const int fold, const double_binned *X, double_complex_binned *Y)</code>	70
2.1.4.109	<code>binned_zbdupdate(const int fold, const double X, double_complex_binned *Y)</code>	70
2.1.4.110	<code>binned_zbnegate(const int fold, double_complex_binned *X)</code>	71
2.1.4.111	<code>binned_zbnum(const int fold)</code>	71
2.1.4.112	<code>binned_zbprint(const int fold, const double_complex_binned *X)</code>	72
2.1.4.113	<code>binned_zbrenorm(const int fold, double_complex_binned *X)</code>	72
2.1.4.114	<code>binned_zbsetzero(const int fold, double_complex_binned *X)</code>	72
2.1.4.115	<code>binned_zbsize(const int fold)</code>	73
2.1.4.116	<code>binned_zbzadd(const int fold, const void *X, double_complex_binned *Y)</code>	73
2.1.4.117	<code>binned_zbzbadd(const int fold, const double_complex_binned *X, double_complex_binned *Y)</code>	74
2.1.4.118	<code>binned_zbzbaddv(const int fold, const int N, const double_complex_binned *X, const int incX, double_complex_binned *Y, const int incY)</code>	74
2.1.4.119	<code>binned_zbzbset(const int fold, const double_complex_binned *X, double_complex_binned *Y)</code>	75
2.1.4.120	<code>binned_zbzconv(const int fold, const void *X, double_complex_binned *Y)</code>	75

2.1.4.121	<code>binned_zbzdeposit(const int fold, const void *X, double_complex_binned *Y)</code>	75
2.1.4.122	<code>binned_zbzupdate(const int fold, const void *X, double_complex_binned *Y)</code>	77
2.1.4.123	<code>binned_zmdenorm(const int fold, const double *priX)</code>	77
2.1.4.124	<code>binned_zmdmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	78
2.1.4.125	<code>binned_zmdrescale(const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	78
2.1.4.126	<code>binned_zmdupdate(const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	79
2.1.4.127	<code>binned_zmnegate(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	79
2.1.4.128	<code>binned_zmprint(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)</code>	80
2.1.4.129	<code>binned_zmrenorm(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	80
2.1.4.130	<code>binned_zmsetzero(const int fold, double *priX, const int incpriX, double *carX, const int inccarX)</code>	81
2.1.4.131	<code>binned_zmzadd(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	81
2.1.4.132	<code>binned_zmzconv(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	82
2.1.4.133	<code>binned_zmzdeposit(const int fold, const void *X, double *priY, const int incpriY)</code>	82
2.1.4.134	<code>binned_zmzmadd(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	83
2.1.4.135	<code>binned_zmzmset(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	83
2.1.4.136	<code>binned_zmzupdate(const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)</code>	84
2.1.4.137	<code>binned_zzbconv_sub(const int fold, const double_complex_binned *X, void *conv)</code>	84
2.1.4.138	<code>binned_zzmconv_sub(const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, void *conv)</code>	85
2.2	<code>include/binnedBLAS.h</code> File Reference	85
2.2.1	Detailed Description	89
2.2.2	Function Documentation	90
2.2.2.1	<code>binnedBLAS_camax_sub(const int N, const void *X, const int incX, void *amax)</code>	90

2.2.2.2	<code>binnedBLAS_camaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	90
2.2.2.3	<code>binnedBLAS_cbcdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_binned *Z)</code>	91
2.2.2.4	<code>binnedBLAS_cbcdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float_binned *Z)</code>	91
2.2.2.5	<code>binnedBLAS_cbcgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, float_complex_binned *C, const int ldc)</code>	92
2.2.2.6	<code>binnedBLAS_cbcgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, float_complex_binned *Y, const int incY)</code>	93
2.2.2.7	<code>binnedBLAS_cbcsum(const int fold, const int N, const void *X, const int incX, float_binned *Y)</code>	93
2.2.2.8	<code>binnedBLAS_cmcdotc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	94
2.2.2.9	<code>binnedBLAS_cmcdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	94
2.2.2.10	<code>binnedBLAS_cmcsu(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	95
2.2.2.11	<code>binnedBLAS_damax(const int N, const double *X, const int incX)</code>	96
2.2.2.12	<code>binnedBLAS_damaxm(const int N, const double *X, const int incX, const double *Y, const int incY)</code>	96
2.2.2.13	<code>binnedBLAS_dbdasum(const int fold, const int N, const double *X, const int incX, double_binned *Y)</code>	97
2.2.2.14	<code>binnedBLAS_dbddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double_binned *Z)</code>	97
2.2.2.15	<code>binnedBLAS_dbdgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, double_binned *C, const int ldc)</code>	98
2.2.2.16	<code>binnedBLAS_dbdgemv(const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, double_binned *Y, const int incY)</code>	99
2.2.2.17	<code>binnedBLAS_dbdssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double_binned *Y)</code>	99
2.2.2.18	<code>binnedBLAS_dbdsum(const int fold, const int N, const double *X, const int incX, double_binned *Y)</code>	100
2.2.2.19	<code>binnedBLAS_dbzasum(const int fold, const int N, const void *X, const int incX, double_binned *Y)</code>	100

2.2.2.20	<code>binnedBLAS_dbzssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double_binned *Y)</code>	101
2.2.2.21	<code>binnedBLAS_dmdasum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	101
2.2.2.22	<code>binnedBLAS_dmddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	102
2.2.2.23	<code>binnedBLAS_dmdssq(const int fold, const int N, const double *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	103
2.2.2.24	<code>binnedBLAS_dmdsum(const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	103
2.2.2.25	<code>binnedBLAS_dmzasum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	104
2.2.2.26	<code>binnedBLAS_dmzssq(const int fold, const int N, const void *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)</code>	104
2.2.2.27	<code>binnedBLAS_samax(const int N, const float *X, const int incX)</code>	105
2.2.2.28	<code>binnedBLAS_samaxm(const int N, const float *X, const int incX, const float *Y, const int incY)</code>	105
2.2.2.29	<code>binnedBLAS_sbcasum(const int fold, const int N, const void *X, const int incX, float_binned *Y)</code>	106
2.2.2.30	<code>binnedBLAS_sbcssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float_binned *Y)</code>	106
2.2.2.31	<code>binnedBLAS_sbsasum(const int fold, const int N, const float *X, const int incX, float_binned *Y)</code>	107
2.2.2.32	<code>binnedBLAS_sbsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float_binned *Z)</code>	107
2.2.2.33	<code>binnedBLAS_sbsgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, float_binned *C, const int ldc)</code>	108
2.2.2.34	<code>binnedBLAS_sbsgemv(const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, float_binned *Y, const int incY)</code>	109
2.2.2.35	<code>binnedBLAS_sbsssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float_binned *Y)</code>	109
2.2.2.36	<code>binnedBLAS_sbssum(const int fold, const int N, const float *X, const int incX, float_binned *Y)</code>	110
2.2.2.37	<code>binnedBLAS_smcasum(const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	110

2.2.2.38	<code>binnedBLAS_smcssq(const int fold, const int N, const void *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	111
2.2.2.39	<code>binnedBLAS_smsasum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	112
2.2.2.40	<code>binnedBLAS_smsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)</code>	112
2.2.2.41	<code>binnedBLAS_smsssq(const int fold, const int N, const float *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)</code>	113
2.2.2.42	<code>binnedBLAS_smssum(const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)</code>	113
2.2.2.43	<code>binnedBLAS_zamax_sub(const int N, const void *X, const int incX, void *amax)</code>	114
2.2.2.44	<code>binnedBLAS_zamaxm_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)</code>	114
2.2.2.45	<code>binnedBLAS_zbzdetc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_binned *Z)</code>	115
2.2.2.46	<code>binnedBLAS_zbzdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double_binned *Z)</code>	115
2.2.2.47	<code>binnedBLAS_zbzgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, double_complex_binned *C, const int ldc)</code>	116
2.2.2.48	<code>binnedBLAS_zbzgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, double_complex_binned *Y, const int incY)</code>	117
2.2.2.49	<code>binnedBLAS_zbzsum(const int fold, const int N, const void *X, const int incX, double_binned *Y)</code>	118
2.2.2.50	<code>binnedBLAS_zmzdetc(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	118
2.2.2.51	<code>binnedBLAS_zmzdotu(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)</code>	119
2.2.2.52	<code>binnedBLAS_zmzsum(const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)</code>	119
2.3	<code>include/binnedMPI.h</code> File Reference	120
2.3.1	Detailed Description	121
2.3.2	Function Documentation	121
2.3.2.1	<code>binnedMPI_CBCBADD(const int fold)</code>	121
2.3.2.2	<code>binnedMPI_DDBBADD(const int fold)</code>	122

2.3.2.3	binnedMPI_DBDBADDSQ(const int fold)	122
2.3.2.4	binnedMPI_DOUBLE_BINNED(const int fold)	123
2.3.2.5	binnedMPI_DOUBLE_BINNED_SCALED(const int fold)	123
2.3.2.6	binnedMPI_DOUBLE_COMPLEX_BINNED(const int fold)	124
2.3.2.7	binnedMPI_FLOAT_BINNED(const int fold)	124
2.3.2.8	binnedMPI_FLOAT_BINNED_SCALED(const int fold)	125
2.3.2.9	binnedMPI_FLOAT_COMPLEX_BINNED(const int fold)	125
2.3.2.10	binnedMPI_SBSBADD(const int fold)	125
2.3.2.11	binnedMPI_SBSBADDSQ(const int fold)	126
2.3.2.12	binnedMPI_ZBZBADD(const int fold)	126
2.4	include/reproBLAS.h File Reference	127
2.4.1	Detailed Description	130
2.4.2	Function Documentation	131
2.4.2.1	reproBLAS_cdotc_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)	131
2.4.2.2	reproBLAS_cdotu_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)	131
2.4.2.3	reproBLAS_cgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)	132
2.4.2.4	reproBLAS_cgmv(const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)	133
2.4.2.5	reproBLAS_csum_sub(const int N, const void *X, const int incX, void *sum)	133
2.4.2.6	reproBLAS_dasum(const int N, const double *X, const int incX)	134
2.4.2.7	reproBLAS_ddot(const int N, const double *X, const int incX, const double *Y, const int incY)	134
2.4.2.8	reproBLAS_dgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)	135
2.4.2.9	reproBLAS_dgmv(const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)	136
2.4.2.10	reproBLAS_dnrm2(const int N, const double *X, const int incX)	137
2.4.2.11	reproBLAS_dsum(const int N, const double *X, const int incX)	137

2.4.2.12	<code>reproBLAS_dzasum(const int N, const void *X, const int incX)</code>	138
2.4.2.13	<code>reproBLAS_dznrm2(const int N, const void *X, int incX)</code>	138
2.4.2.14	<code>reproBLAS_rcdotc_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)</code>	139
2.4.2.15	<code>reproBLAS_rcdotu_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)</code>	139
2.4.2.16	<code>reproBLAS_rcgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)</code>	140
2.4.2.17	<code>reproBLAS_rcgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)</code>	141
2.4.2.18	<code>reproBLAS_rcsum_sub(const int fold, const int N, const void *X, const int incX, void *sum)</code>	142
2.4.2.19	<code>reproBLAS_rdasum(const int fold, const int N, const double *X, const int incX)</code>	142
2.4.2.20	<code>reproBLAS_rddot(const int fold, const int N, const double *X, const int incX, const double *Y, const int incY)</code>	143
2.4.2.21	<code>reproBLAS_rdgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)</code>	143
2.4.2.22	<code>reproBLAS_rdgemv(const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)</code>	144
2.4.2.23	<code>reproBLAS_rdnrm2(const int fold, const int N, const double *X, const int incX)</code>	145
2.4.2.24	<code>reproBLAS_rdsun(const int fold, const int N, const double *X, const int incX)</code>	146
2.4.2.25	<code>reproBLAS_rdzasum(const int fold, const int N, const void *X, const int incX)</code>	146
2.4.2.26	<code>reproBLAS_rdznrm2(const int fold, const int N, const void *X, int incX)</code>	147
2.4.2.27	<code>reproBLAS_rsasum(const int fold, const int N, const float *X, const int incX)</code>	148
2.4.2.28	<code>reproBLAS_rscasum(const int fold, const int N, const void *X, const int incX)</code>	149
2.4.2.29	<code>reproBLAS_rscnrm2(const int fold, const int N, const void *X, const int incX)</code>	150
2.4.2.30	<code>reproBLAS_rsdot(const int fold, const int N, const float *X, const int incX, const float *Y, const int incY)</code>	151
2.4.2.31	<code>reproBLAS_rsgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)</code>	152

2.4.2.32	<code>reproBLAS_rsgemv(const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)</code>	153
2.4.2.33	<code>reproBLAS_rsnrm2(const int fold, const int N, const float *X, const int incX)</code>	153
2.4.2.34	<code>reproBLAS_rssum(const int fold, const int N, const float *X, const int incX)</code>	154
2.4.2.35	<code>reproBLAS_rzdotc_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)</code>	154
2.4.2.36	<code>reproBLAS_rzdotu_sub(const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)</code>	155
2.4.2.37	<code>reproBLAS_rzgemm(const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)</code>	156
2.4.2.38	<code>reproBLAS_rzgemv(const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)</code>	157
2.4.2.39	<code>reproBLAS_rzsum_sub(const int fold, const int N, const void *X, const int incX, void *sum)</code>	157
2.4.2.40	<code>reproBLAS_sasum(const int N, const float *X, const int incX)</code>	158
2.4.2.41	<code>reproBLAS_scasum(const int N, const void *X, const int incX)</code>	158
2.4.2.42	<code>reproBLAS_scnrm2(const int N, const void *X, const int incX)</code>	159
2.4.2.43	<code>reproBLAS_sdot(const int N, const float *X, const int incX, const float *Y, const int incY)</code>	159
2.4.2.44	<code>reproBLAS_sgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)</code>	160
2.4.2.45	<code>reproBLAS_sgemv(const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)</code>	161
2.4.2.46	<code>reproBLAS_snrm2(const int N, const float *X, const int incX)</code>	161
2.4.2.47	<code>reproBLAS_ssum(const int N, const float *X, const int incX)</code>	162
2.4.2.48	<code>reproBLAS_zdotc_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)</code>	162
2.4.2.49	<code>reproBLAS_zdotu_sub(const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)</code>	163
2.4.2.50	<code>reproBLAS_zgemm(const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)</code>	163
2.4.2.51	<code>reproBLAS_zgemv(const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)</code>	164
2.4.2.52	<code>reproBLAS_zsum_sub(const int N, const void *X, const int incX, void *sum)</code>	166

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

include/ binned.h	
Binned.h defines the binned types and the lower level functions associated with their use . . .	3
include/ binnedBLAS.h	
BinnedBLAS.h defines BLAS Methods that operate on binned types	85
include/ binnedMPI.h	
BinnedMPI.h defines MPI wrapper functions for binned types and the necessary functions to perform reproducible reductions	120
include/ reproBLAS.h	
ReproBLAS.h defines reproducible BLAS Methods	127

Chapter 2

File Documentation

2.1 include/binned.h File Reference

[binned.h](#) defines the binned types and the lower level functions associated with their use.

```
#include <stddef.h>
#include <stdlib.h>
#include <float.h>
```

Macros

- `#define DBWIDTH 40`
Binned double precision bin width.
- `#define SBWIDTH 13`
Binned single precision bin width.
- `#define binned_DBMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DBWIDTH) - 1)`
Binned double precision maximum index.
- `#define binned_SBMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SBWIDTH) - 1)`
Binned single precision maximum index.
- `#define binned_DBMAXFOLD (binned_DBMAXINDEX + 1)`
The maximum double precision fold supported by the library.
- `#define binned_SBMAXFOLD (binned_SBMAXINDEX + 1)`
The maximum single precision fold supported by the library.
- `#define binned_DBENDURANCE (1 << (DBL_MANT_DIG - DBWIDTH - 2))`
Binned double precision deposit endurance.
- `#define binned_SBENDURANCE (1 << (FLT_MANT_DIG - SBWIDTH - 2))`
Binned single precision deposit endurance.
- `#define binned_DBCAPACITY (binned_DBENDURANCE*(1.0/DBL_EPSILON - 1.0))`
Binned double precision capacity.
- `#define binned_SBCAPACITY (binned_SBENDURANCE*(1.0/FLT_EPSILON - 1.0))`
Binned single precision capacity.
- `#define binned_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DBWIDTH + 1)))`
Binned double precision compression factor.
- `#define binned_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SBWIDTH + 1)))`
Binned single precision compression factor.
- `#define binned_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DBWIDTH + 1)))`
Binned double precision expansion factor.
- `#define binned_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SBWIDTH + 1)))`
Binned single precision expansion factor.

Typedefs

- typedef double [double_binned](#)
The binned double datatype.
- typedef double [double_complex_binned](#)
The binned complex double datatype.
- typedef float [float_binned](#)
The binned float datatype.
- typedef float [float_complex_binned](#)
The binned complex float datatype.

Functions

- size_t [binned_dbsize](#) (const int fold)
binned double precision size
- size_t [binned_zbsize](#) (const int fold)
binned complex double precision size
- size_t [binned_sbsize](#) (const int fold)
binned single precision size
- size_t [binned_cbsize](#) (const int fold)
binned complex single precision size
- [double_binned](#) * [binned_dballoc](#) (const int fold)
binned double precision allocation
- [double_complex_binned](#) * [binned_zballoc](#) (const int fold)
binned complex double precision allocation
- [float_binned](#) * [binned_sballoc](#) (const int fold)
binned single precision allocation
- [float_complex_binned](#) * [binned_cballoc](#) (const int fold)
binned complex single precision allocation
- int [binned_dbnum](#) (const int fold)
binned double precision size
- int [binned_zbnum](#) (const int fold)
binned complex double precision size
- int [binned_sbnum](#) (const int fold)
binned single precision size
- int [binned_cbnum](#) (const int fold)
binned complex single precision size
- double [binned_dbbound](#) (const int fold, const int N, const double X, const double S)
Get binned double precision summation error bound.
- float [binned_sbbound](#) (const int fold, const int N, const float X, const float S)
Get binned single precision summation error bound.
- const double * [binned_dmbins](#) (const int X)
Get binned double precision reference bins.
- const float * [binned_smbins](#) (const int X)
Get binned single precision reference bins.
- int [binned_dindex](#) (const double X)
Get index of double precision.
- int [binned_dmindex](#) (const double *priX)
Get index of manually specified binned double precision.
- int [binned_dmindex0](#) (const double *priX)

- Check if index of manually specified binned double precision is 0.*

 - int `binned_sindex` (const float X)

Get index of single precision.
- int `binned_sminindex` (const float *priX)

Get index of manually specified binned single precision.
- int `binned_sminindex0` (const float *priX)

Check if index of manually specified binned single precision is 0.
- int `binned_dmdenorm` (const int fold, const double *priX)

Check if binned type has denormal bits.
- int `binned_zmdenorm` (const int fold, const double *priX)

Check if binned type has denormal bits.
- int `binned_smdenorm` (const int fold, const float *priX)

Check if binned type has denormal bits.
- int `binned_cmndenorm` (const int fold, const float *priX)

Check if binned type has denormal bits.
- void `binned_dbprint` (const int fold, const `double_binned` *X)

Print binned double precision.
- void `binned_dmprint` (const int fold, const double *priX, const int incpriX, const double *carX, const int incarX)

Print manually specified binned double precision.
- void `binned_zbprint` (const int fold, const `double_complex_binned` *X)

Print binned complex double precision.
- void `binned_zmprint` (const int fold, const double *priX, const int incpriX, const double *carX, const int incarX)

Print manually specified binned complex double precision.
- void `binned_sbprint` (const int fold, const `float_binned` *X)

Print binned single precision.
- void `binned_smprint` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)

Print manually specified binned single precision.
- void `binned_cbprint` (const int fold, const `float_complex_binned` *X)

Print binned complex single precision.
- void `binned_cmprint` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)

Print manually specified binned complex single precision.
- void `binned_dbdbset` (const int fold, const `double_binned` *X, `double_binned` *Y)

Set binned double precision (Y = X)
- void `binned_dmdmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

Set manually specified binned double precision (Y = X)
- void `binned_zbzbset` (const int fold, const `double_complex_binned` *X, `double_complex_binned` *Y)

Set binned complex double precision (Y = X)
- void `binned_zmzmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

Set manually specified binned complex double precision (Y = X)
- void `binned_zbdbset` (const int fold, const `double_binned` *X, `double_complex_binned` *Y)

Set binned complex double precision to binned double precision (Y = X)
- void `binned_zmdmset` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)

Set manually specified binned complex double precision to manually specified binned double precision (Y = X)
- void `binned_sbsbset` (const int fold, const `float_binned` *X, `float_binned` *Y)

Set binned single precision (Y = X)

- void `binned_smsmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Set manually specified binned single precision ($Y = X$)
- void `binned_cbcbsset` (const int fold, const `float_complex_binned` *X, `float_complex_binned` *Y)
Set binned complex single precision ($Y = X$)
- void `binned_cmcmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Set manually specified binned complex single precision ($Y = X$)
- void `binned_cbsbset` (const int fold, const `float_binned` *X, `float_complex_binned` *Y)
Set binned complex single precision to binned single precision ($Y = X$)
- void `binned_cmsmset` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Set manually specified binned complex single precision to manually specified binned single precision ($Y = X$)
- void `binned_dbsetzero` (const int fold, `double_binned` *X)
Set binned double precision to 0 ($X = 0$)
- void `binned_dmsetzero` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Set manually specified binned double precision to 0 ($X = 0$)
- void `binned_zbsetzero` (const int fold, `double_complex_binned` *X)
Set binned double precision to 0 ($X = 0$)
- void `binned_zmsetzero` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Set manually specified binned complex double precision to 0 ($X = 0$)
- void `binned_sbsetzero` (const int fold, `float_binned` *X)
Set binned single precision to 0 ($X = 0$)
- void `binned_smsetzero` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Set manually specified binned single precision to 0 ($X = 0$)
- void `binned_cbsetzero` (const int fold, `float_complex_binned` *X)
Set binned single precision to 0 ($X = 0$)
- void `binned_cmsetzero` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Set manually specified binned complex single precision to 0 ($X = 0$)
- void `binned_dbdbadd` (const int fold, const `double_binned` *X, `double_binned` *Y)
Add binned double precision ($Y += X$)
- void `binned_dmdmadd` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
Add manually specified binned double precision ($Y += X$)
- void `binned_zbzbadd` (const int fold, const `double_complex_binned` *X, `double_complex_binned` *Y)
Add binned complex double precision ($Y += X$)
- void `binned_zmzmadd` (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, double *priY, const int incpriY, double *carY, const int inccarY)
Add manually specified binned complex double precision ($Y += X$)
- void `binned_sbsbadd` (const int fold, const `float_binned` *X, `float_binned` *Y)
Add binned single precision ($Y += X$)
- void `binned_smsmadd` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Add manually specified binned single precision ($Y += X$)
- void `binned_cbcbsbadd` (const int fold, const `float_complex_binned` *X, `float_complex_binned` *Y)
Add binned complex single precision ($Y += X$)
- void `binned_cmcmadd` (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, float *priY, const int incpriY, float *carY, const int inccarY)
Add manually specified binned complex single precision ($Y += X$)
- void `binned_dbdbaddv` (const int fold, const int N, const `double_binned` *X, const int incX, `double_binned` *Y, const int incY)
Add binned double precision vectors ($Y += X$)

- void `binned_zbzbaddv` (const int fold, const int N, const `double_complex_binned` *X, const int incX, `double_complex_binned` *Y, const int incY)
Add binned complex double precision vectors (Y += X)
- void `binned_sbsbaddv` (const int fold, const int N, const `float_binned` *X, const int incX, `float_binned` *Y, const int incY)
Add binned single precision vectors (Y += X)
- void `binned_cbcdbaddv` (const int fold, const int N, const `float_complex_binned` *X, const int incX, `float_complex_binned` *Y, const int incY)
Add binned complex single precision vectors (Y += X)
- void `binned_dbdadd` (const int fold, const double X, `double_binned` *Y)
Add double precision to binned double precision (Y += X)
- void `binned_dmdadd` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Add double precision to manually specified binned double precision (Y += X)
- void `binned_zbzadd` (const int fold, const void *X, `double_complex_binned` *Y)
Add complex double precision to binned complex double precision (Y += X)
- void `binned_zmzadd` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
Add complex double precision to manually specified binned complex double precision (Y += X)
- void `binned_sbsadd` (const int fold, const float X, `float_binned` *Y)
Add single precision to binned single precision (Y += X)
- void `binned_smsadd` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
Add single precision to manually specified binned single precision (Y += X)
- void `binned_cbcadd` (const int fold, const void *X, `float_complex_binned` *Y)
Add complex single precision to binned complex single precision (Y += X)
- void `binned_cmcadd` (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
Add complex single precision to manually specified binned complex single precision (Y += X)
- void `binned_dbdupdate` (const int fold, const double X, `double_binned` *Y)
Update binned double precision with double precision (X -> Y)
- void `binned_dmdupdate` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified binned double precision with double precision (X -> Y)
- void `binned_zbzupdate` (const int fold, const void *X, `double_complex_binned` *Y)
Update binned complex double precision with complex double precision (X -> Y)
- void `binned_zmzupdate` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified binned complex double precision with complex double precision (X -> Y)
- void `binned_zbdupdate` (const int fold, const double X, `double_complex_binned` *Y)
Update binned complex double precision with double precision (X -> Y)
- void `binned_zmdupdate` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
Update manually specified binned complex double precision with double precision (X -> Y)
- void `binned_sbsupdate` (const int fold, const float X, `float_binned` *Y)
Update binned single precision with single precision (X -> Y)
- void `binned_smsupdate` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
Update manually specified binned single precision with single precision (X -> Y)
- void `binned_cbcupdate` (const int fold, const void *X, `float_complex_binned` *Y)
Update binned complex single precision with complex single precision (X -> Y)
- void `binned_cmcupdate` (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
Update manually specified binned complex single precision with complex single precision (X -> Y)
- void `binned_cbsupdate` (const int fold, const float X, `float_complex_binned` *Y)

- Update binned complex single precision with single precision ($X \rightarrow Y$)*

 - void `binned_cmsupdate` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
- Update manually specified binned complex single precision with single precision ($X \rightarrow Y$)*

 - void `binned_dbddeposit` (const int fold, const double X, `double_binned` *Y)
- Add double precision to suitably binned binned double precision ($Y += X$)*

 - void `binned_dmddeposit` (const int fold, const double X, double *priY, const int incpriY)
- Add double precision to suitably binned manually specified binned double precision ($Y += X$)*

 - void `binned_zbzddeposit` (const int fold, const void *X, `double_complex_binned` *Y)
- Add complex double precision to suitably binned binned complex double precision ($Y += X$)*

 - void `binned_zmzdeposit` (const int fold, const void *X, double *priY, const int incpriY)
- Add complex double precision to suitably binned manually specified binned complex double precision ($Y += X$)*

 - void `binned_sbsdeposit` (const int fold, const float X, `float_binned` *Y)
- Add single precision to suitably binned binned single precision ($Y += X$)*

 - void `binned_smsdeposit` (const int fold, const float X, float *priY, const int incpriY)
- Add single precision to suitably binned manually specified binned single precision ($Y += X$)*

 - void `binned_cbcdeposit` (const int fold, const void *X, `float_complex_binned` *Y)
- Add complex single precision to suitably binned binned complex single precision ($Y += X$)*

 - void `binned_cmcddeposit` (const int fold, const void *X, float *priY, const int incpriY)
- Add complex single precision to suitably binned manually specified binned complex single precision ($Y += X$)*

 - void `binned_dbrenorm` (const int fold, `double_binned` *X)
- Renormalize binned double precision.*

 - void `binned_dmrenorm` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Renormalize manually specified binned double precision.*

 - void `binned_zbrenorm` (const int fold, `double_complex_binned` *X)
- Renormalize binned complex double precision.*

 - void `binned_zmrenorm` (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
- Renormalize manually specified binned complex double precision.*

 - void `binned_sbrenorm` (const int fold, `float_binned` *X)
- Renormalize binned single precision.*

 - void `binned_smrenorm` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Renormalize manually specified binned single precision.*

 - void `binned_cbrenorm` (const int fold, `float_complex_binned` *X)
- Renormalize binned complex single precision.*

 - void `binned_cmrenorm` (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
- Renormalize manually specified binned complex single precision.*

 - void `binned_dbdconv` (const int fold, const double X, `double_binned` *Y)
- Convert double precision to binned double precision ($X \rightarrow Y$)*

 - void `binned_dmdconv` (const int fold, const double X, double *priY, const int incpriY, double *carY, const int inccarY)
- Convert double precision to manually specified binned double precision ($X \rightarrow Y$)*

 - void `binned_zbzconv` (const int fold, const void *X, `double_complex_binned` *Y)
- Convert complex double precision to binned complex double precision ($X \rightarrow Y$)*

 - void `binned_zmzconv` (const int fold, const void *X, double *priY, const int incpriY, double *carY, const int inccarY)
- Convert complex double precision to manually specified binned complex double precision ($X \rightarrow Y$)*

 - void `binned_sbsconv` (const int fold, const float X, `float_binned` *Y)
- Convert single precision to binned single precision ($X \rightarrow Y$)*

 - void `binned_smsconv` (const int fold, const float X, float *priY, const int incpriY, float *carY, const int inccarY)
- Convert single precision to manually specified binned single precision ($X \rightarrow Y$)*

 - void `binned_cbconv` (const int fold, const void *X, `float_complex_binned` *Y)
- Convert complex single precision to binned complex single precision ($X \rightarrow Y$)*

- void [binned_cmconv](#) (const int fold, const void *X, float *priY, const int incpriY, float *carY, const int inccarY)
Convert complex single precision to manually specified binned complex single precision ($X \rightarrow Y$)
- double [binned_ddbconv](#) (const int fold, const [double_binned](#) *X)
Convert binned double precision to double precision ($X \rightarrow Y$)
- double [binned_ddmconv](#) (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX)
Convert manually specified binned double precision to double precision ($X \rightarrow Y$)
- void [binned_zzbconv_sub](#) (const int fold, const [double_complex_binned](#) *X, void *conv)
Convert binned complex double precision to complex double precision ($X \rightarrow Y$)
- void [binned_zzmconv_sub](#) (const int fold, const double *priX, const int incpriX, const double *carX, const int inccarX, void *conv)
Convert manually specified binned complex double precision to complex double precision ($X \rightarrow Y$)
- float [binned_ssbconv](#) (const int fold, const [float_binned](#) *X)
Convert binned single precision to single precision ($X \rightarrow Y$)
- float [binned_ssmconv](#) (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX)
Convert manually specified binned single precision to single precision ($X \rightarrow Y$)
- void [binned_ccbconv_sub](#) (const int fold, const [float_complex_binned](#) *X, void *conv)
Convert binned complex single precision to complex single precision ($X \rightarrow Y$)
- void [binned_ccmconv_sub](#) (const int fold, const float *priX, const int incpriX, const float *carX, const int inccarX, void *conv)
Convert manually specified binned complex single precision to complex single precision ($X \rightarrow Y$)
- void [binned_dbnegate](#) (const int fold, [double_binned](#) *X)
Negate binned double precision ($X = -X$)
- void [binned_dmnegate](#) (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Negate manually specified binned double precision ($X = -X$)
- void [binned_zbnegate](#) (const int fold, [double_complex_binned](#) *X)
Negate binned complex double precision ($X = -X$)
- void [binned_zmnegate](#) (const int fold, double *priX, const int incpriX, double *carX, const int inccarX)
Negate manually specified binned complex double precision ($X = -X$)
- void [binned_sbnegate](#) (const int fold, [float_binned](#) *X)
Negate binned single precision ($X = -X$)
- void [binned_smnegate](#) (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Negate manually specified binned single precision ($X = -X$)
- void [binned_cbnegate](#) (const int fold, [float_complex_binned](#) *X)
Negate binned complex single precision ($X = -X$)
- void [binned_cmnegate](#) (const int fold, float *priX, const int incpriX, float *carX, const int inccarX)
Negate manually specified binned complex single precision ($X = -X$)
- double [binned_dscale](#) (const double X)
Get a reproducible double precision scale.
- float [binned_sscale](#) (const float X)
Get a reproducible single precision scale.
- void [binned_dmdrescale](#) (const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
rescale manually specified binned double precision sum of squares
- void [binned_zmdrescale](#) (const int fold, const double X, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
rescale manually specified binned complex double precision sum of squares
- void [binned_smsrescale](#) (const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)
rescale manually specified binned single precision sum of squares
- void [binned_cmsrescale](#) (const int fold, const float X, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)

rescale manually specified binned complex single precision sum of squares

- double `binned_dmdmaddsq` (const int fold, const double scaleX, const double *priX, const int incpriX, const double *carX, const int inccarX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)

Add manually specified binned double precision scaled sums of squares (Y += X)

- double `binned_dbdbaddsq` (const int fold, const double scaleX, const `double_binned` *X, const double scaleY, `double_binned` *Y)

Add binned double precision scaled sums of squares (Y += X)

- float `binned_smsmaddsq` (const int fold, const float scaleX, const float *priX, const int incpriX, const float *carX, const int inccarX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)

Add manually specified binned single precision scaled sums of squares (Y += X)

- float `binned_sbsbaddsq` (const int fold, const float scaleX, const `float_binned` *X, const float scaleY, `float_binned` *Y)

Add binned single precision scaled sums of squares (Y += X)

- double `binned_uftp` (const double X)

unit in the first place

- float `binned_uftp` (const float X)

unit in the first place

2.1.1 Detailed Description

`binned.h` defines the binned types and the lower level functions associated with their use.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfop` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double (`double_binned`)
- zb - binned complex double (`double_complex_binned`)
- sb - binned float (`float_binned`)
- cb - binned complex float (`float_complex_binned`)
- dm - manually specified binned double (`double, double`)
- zm - manually specified binned complex double (`double, double`)
- sm - manually specified binned float (`float, float`)
- cm - manually specified binned complex float (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by `sub`, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in `config.h`. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an binned type themselves. If you do not know what these are, don't worry about the manually specified binned types.

2.1.2 Macro Definition Documentation

2.1.2.1 `#define binned_DBCAPACITY (binned_DBENDURANCE*(1.0/DBL_EPSILON - 1.0))`

Binned double precision capacity.

The maximum number of double precision numbers that can be summed using binned double precision. Applies also to binned complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.2 `#define binned_DBENDURANCE (1 << (DBL_MANT_DIG - DBWIDTH - 2))`

Binned double precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to binned complex double precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.3 `#define binned_DBMAXFOLD (binned_DBMAXINDEX + 1)`

The maximum double precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.4 `#define binned_DBMAXINDEX (((DBL_MAX_EXP - DBL_MIN_EXP + DBL_MANT_DIG - 1)/DBWIDTH) - 1)`

Binned double precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.5 `#define binned_DMCOMPRESSION (1.0/(1 << (DBL_MANT_DIG - DBWIDTH + 1)))`

Binned double precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.6 `#define binned_DMEXPANSION (1.0*(1 << (DBL_MANT_DIG - DBWIDTH + 1)))`

Binned double precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.7 `#define binned_SBCAPACITY (binned_SBENDURANCE*(1.0/FLT_EPSILON - 1.0))`

Binned single precision capacity.

The maximum number of single precision numbers that can be summed using binned single precision. Applies also to binned complex double precision.

Author

Peter Ahrens

Date

27 Apr 2015

2.1.2.8 `#define binned_SBENDURANCE (1 << (FLT_MANT_DIG - SBWIDTH - 2))`

Binned single precision deposit endurance.

The number of deposits that can be performed before a renorm is necessary. Applies also to binned complex single precision.

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.9 `#define binned_SBMAXFOLD (binned_SBMAXINDEX + 1)`

The maximum single precision fold supported by the library.

Author

Peter Ahrens

Date

14 Jan 2016

2.1.2.10 `#define binned_SBMAXINDEX (((FLT_MAX_EXP - FLT_MIN_EXP + FLT_MANT_DIG - 1)/SBWIDTH) - 1)`

Binned single precision maximum index.

maximum index (inclusive)

Author

Peter Ahrens

Date

24 Jun 2015

2.1.2.11 `#define binned_SMCOMPRESSION (1.0/(1 << (FLT_MANT_DIG - SBWIDTH + 1)))`

Binned single precision compression factor.

This factor is used to scale down inputs before deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.12 `#define binned_SMEXPANSION (1.0*(1 << (FLT_MANT_DIG - SBWIDTH + 1)))`

Binned single precision expansion factor.

This factor is used to scale up inputs after deposition into the bin of highest index

Author

Peter Ahrens

Date

19 May 2015

2.1.2.13 #define DBWIDTH 40

Binned double precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.2.14 #define SBWIDTH 13

Binned single precision bin width.

bin width (in bits)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.3 Typedef Documentation

2.1.3.1 typedef double double_binned

The binned double datatype.

To allocate a [double_binned](#), call [binned_dballoc\(\)](#)

Warning

A [double_binned](#) is, under the hood, an array of `double`. Therefore, if you have defined an array of [double_binned](#), you must index it by multiplying the index into the array by the number of underlying `double` that make up the [double_binned](#). This number can be obtained by a call to [binned_dbnum\(\)](#)

2.1.3.2 typedef double double_complex_binned

The binned complex double datatype.

To allocate a `double_complex_binned`, call `binned_zballoc()`

Warning

A `double_complex_binned` is, under the hood, an array of `double`. Therefore, if you have defined an array of `double_complex_binned`, you must index it by multiplying the index into the array by the number of underlying `double` that make up the `double_complex_binned`. This number can be obtained by a call to `binned_zbnum()`

2.1.3.3 typedef float float_binned

The binned float datatype.

To allocate a `float_binned`, call `binned_sballot()`

Warning

A `float_binned` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_binned`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_binned`. This number can be obtained by a call to `binned_sbnun()`

2.1.3.4 typedef float float_complex_binned

The binned complex float datatype.

To allocate a `float_complex_binned`, call `binned_cballot()`

Warning

A `float_complex_binned` is, under the hood, an array of `float`. Therefore, if you have defined an array of `float_complex_binned`, you must index it by multiplying the index into the array by the number of underlying `float` that make up the `float_complex_binned`. This number can be obtained by a call to `binned_cbnun()`

2.1.4 Function Documentation

2.1.4.1 float_complex_binned* binned_cballot (const int *fold*)

binned complex single precision allocation

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.2 void binned_cbcadd (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Add complex single precision to binned complex single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.3 void binned_cbcadd (const int *fold*, const float_complex_binned * *X*, float_complex_binned * *Y*)

Add binned complex single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.4 void `binned_cbcaddv` (const int *fold*, const int *N*, const float_complex_binned * *X*, const int *incX*, float_complex_binned * *Y*, const int *incY*)

Add binned complex single precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	binned vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.5 void `binned_cbcset` (const int *fold*, const float_complex_binned * *X*, float_complex_binned * *Y*)

Set binned complex single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.6 void binned_cbconv (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Convert complex single precision to binned complex single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.7 void binned_cbcdeposit (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Add complex single precision to suitably binned binned complex single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [binned_cbcupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [binned_cbcdeposit\(\)](#) to deposit a maximum of [binned_SBENDURANCE](#) elements into *Y* before renormalizing *Y* with [binned_cbrenorm\(\)](#). After any number of successive calls of [binned_cbcdeposit\(\)](#) on *Y*, you must renormalize *Y* with [binned_cbrenorm\(\)](#) before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.8 void `binned_cbcupdate` (const int *fold*, const void * *X*, float_complex_binned * *Y*)

Update binned complex single precision with complex single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.9 void `binned_cbnegate` (const int *fold*, float_complex_binned * *X*)

Negate binned complex single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.10 int `binned_cbnum` (const int *fold*)

binned complex single precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in `float`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.11 `void binned_cbprint (const int fold, const float_complex_binned * X)`

Print binned complex single precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.12 `void binned_cbrenorm (const int fold, float_complex_binned * X)`

Renormalize binned complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.13 `void binned_cbsbset (const int fold, const float_binned * X, float_complex_binned * Y)`

Set binned complex single precision to binned single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.14 `void binned_cbsetzero (const int fold, float_complex_binned * X)`

Set binned single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.15 `size_t binned_cbsize (const int fold)`

binned complex single precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.16 `void binned_cbsupdate (const int fold, const float X, float_complex_binned * Y)`

Update binned complex single precision with single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.17 `void binned_ccbconv_sub (const int fold, const float_complex_binned * X, void * conv)`

Convert binned complex single precision to complex single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.18 `void binned_ccmconv_sub (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, void * conv)`

Convert manually specified binned complex single precision to complex single precision (X -> Y)

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.19 `void binned_cmcadd (const int fold, const void * X, float * priY, const int incpriY, float * carY, const int inccarY)`

Add complex single precision to manually specified binned complex single precision (Y += X)

Performs the operation Y += X on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.20 void `binned_cmconv` (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Convert complex single precision to manually specified binned complex single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	<i>Y</i> 's carry vector
<i>inccarY</i>	stride within <i>Y</i> 's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.21 void `binned_cmcddeposit` (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*)

Add complex single precision to suitably binned manually specified binned complex single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called `binned_cmupdate()` on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call `binned_cmcddeposit()` to deposit a maximum of `binned_SBENDURANCE` elements into *Y* before renormalizing *Y* with `binned_cmrenorm()`. After any number of successive calls of `binned_cmcddeposit()` on *Y*, you must renormalize *Y* with `binned_cmrenorm()` before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.22 `void binned_cmcmadd (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, float * priY, const int incpriY, float * carY, const int inccarY)`

Add manually specified binned complex single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.23 `void binned_cmcmset (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, float * priY, const int incpriY, float * carY, const int inccarY)`

Set manually specified binned complex single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)

Parameters

<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.24 void binned_cmcupdate (const int *fold*, const void * *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Update manually specified binned complex single precision with complex single precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.25 int binned_cmdenorm (const int *fold*, const float * *priX*)

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the binned type
<i>priX</i>	X's primary vector

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.26 void binned_cmnegate (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Negate manually specified binned complex single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.27 void binned_cmprint (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*)

Print manually specified binned complex single precision.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Parameters

<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.28 void binned_cmrenorm (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified binned complex single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.29 void binned_cmsetzero (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Set manually specified binned complex single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.30 void `binned_cmsmset` (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Set manually specified binned complex single precision to manually specified binned single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.31 void `binned_cmsrescale` (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified binned complex single precision sum of squares

Rescale an binned complex single precision sum of squares Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	Y's new scaleY ($X == \text{binned_sscale}(f)$ for some <code>float f</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{binned_sscale}(f)$ for some <code>float f</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.32 `void binned_cmsupdate (const int fold, const float X, float * priY, const int incpriY, float * carY, const int inccarY)`

Update manually specified binned complex single precision with single precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.33 `double_binned* binned_dballoc (const int fold)`

binned double precision allocation

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.34 `double binned_dbbound (const int fold, const int N, const double X, const double S)`

Get binned double precision summation error bound.

This is a bound on the absolute error of a summation using binned types

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	the number of double precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using binned types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.35 `void binned_dbdadd (const int fold, const double X, double_binned * Y)`

Add double precision to binned double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.36 `void binned_dbdbadd (const int fold, const double_binned * X, double_binned * Y)`

Add binned double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.37 `double binned_dbdbaddsq (const int fold, const double scaleX, const double_binned * X, const double scaleY, double_binned * Y)`

Add binned double precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the binned types
<i>scaleX</i>	scale of X ($\text{scaleX} == \text{binned_dscale}(Z)$ for some double Z)
<i>X</i>	binned scalar X
<i>scaleY</i>	scale of Y ($\text{scaleY} == \text{binned_dscale}(Z)$ for some double Z)
<i>Y</i>	binned scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.38 void `binned_dbdbaddv` (const int *fold*, const int *N*, const double_binned * *X*, const int *incX*, double_binned * *Y*, const int *incY*)

Add binned double precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	binned vector X
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned vector Y
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.39 void `binned_dbdbset` (const int *fold*, const double_binned * *X*, double_binned * *Y*)

Set binned double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.40 `void binned_dbdconv (const int fold, const double X, double_binned * Y)`

Convert double precision to binned double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.41 `void binned_dbddeposit (const int fold, const double X, double_binned * Y)`

Add double precision to suitably binned binned double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [binned_dbdupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [binned_dbddeposit\(\)](#) to deposit a maximum of [binned_DBENDURANCE](#) elements into *Y* before renormalizing *Y* with [binned_dbrenorm\(\)](#). After any number of successive calls of [binned_dbddeposit\(\)](#) on *Y*, you must renormalize *Y* with [binned_dbrenorm\(\)](#) before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.42 void binned_dbdupdate (const int *fold*, const double *X*, double_binned * *Y*)

Update binned double precision with double precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.43 void binned_dbnegate (const int *fold*, double_binned * *X*)

Negate binned double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.44 `int binned_dbnum (const int fold)`

binned double precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in `double`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.45 `void binned_dbprint (const int fold, const double_binned * X)`

Print binned double precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.46 `void binned_dbrenorm (const int fold, double_binned * X)`

Renormalize binned double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.47 void binned_dbsetzero (const int *fold*, double_binned * *X*)

Set binned double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.48 size_t binned_dbsize (const int *fold*)

binned double precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.49 `double binned_ddbconv (const int fold, const double_binned * X)`

Convert binned double precision to double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.50 `double binned_ddmconv (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX)`

Convert manually specified binned double precision to double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Returns

scalar Y

Author

Peter Ahrens

Date

31 Jul 2015

2.1.4.51 int binned_dindex (const double *X*)

Get index of double precision.

The index of a non-binned type is the smallest index an binned type would need to have to sum it reproducibly. Higher indicies correspond to smaller bins.

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.52 const double* binned_dmbins (const int *X*)

Get binned double precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

<i>X</i>	index
----------	-------

Returns

pointer to constant double precision bins of index *X*

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.53 void binned_dmdadd (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add double precision to manually specified binned double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y 's primary vector
<i>incpriY</i>	stride within Y 's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y 's carry vector
<i>inccarY</i>	stride within Y 's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.54 void binned_dmdconv (const int *fold*, const double *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Convert double precision to manually specified binned double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y 's primary vector
<i>incpriY</i>	stride within Y 's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y 's carry vector
<i>inccarY</i>	stride within Y 's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.55 void `binned_dmddeposit (const int fold, const double X, double * priY, const int incpriY)`

Add double precision to suitably binned manually specified binned double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y where the index of Y is larger than the index of X

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called `binned_dmdupdate()` on Y with the maximum absolute value of all future elements you wish to deposit in Y , you can call `binned_dmddeposit()` to deposit a maximum of `binned_DBENDURANCE` elements into Y before renormalizing Y with `binned_dmrenorm()`. After any number of successive calls of `binned_dmddeposit()` on Y , you must renormalize Y with `binned_dmrenorm()` before using any other function on Y .

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y 's primary vector
<i>incpriY</i>	stride within Y 's primary vector (use every $incpriY$ 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.56 int `binned_dmdenorm (const int fold, const double * priX)`

Check if binned type has denormal bits.

A quick check to determine if calculations involving X cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the binned type
<i>priX</i>	X 's primary vector

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.57 `void binned_dmdmadd (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX, double * priY, const int incpriY, double * carY, const int inccarY)`

Add manually specified binned double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.58 `double binned_dmdmaddsq (const int fold, const double scaleX, const double * priX, const int incpriX, const double * carX, const int inccarX, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY)`

Add manually specified binned double precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the binned types
<i>scaleX</i>	scale of X (<i>scaleX</i> == binned_dscale (Z) for some <code>double Z</code>)
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>scaleY</i>	scale of Y (<i>scaleY</i> == binned_dscale (Z) for some <code>double Z</code>)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.59 void binmed_dmdmset (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Set manually specified binmed double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binmed types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.60 void binmed_dmdrescale (const int *fold*, const double *X*, const double *scaleY*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

rescale manually specified binmed double precision sum of squares

Rescale an binmed double precision sum of squares Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	Y's new scaleY ($X == \text{binned_dscale}(f)$ for some <code>double f</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{binned_dscale}(f)$ for some <code>double f</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.61 `void binned_dmdupdate (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Update manually specified binned double precision with double precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.62 `int binned_dmindex (const double * priX)`

Get index of manually specified binned double precision.

The index of an binned type is the bin that it corresponds to. Higher indicies correspond to smaller bins.

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.63 int binned_dmindex0 (const double * *priX*)

Check if index of manually specified binned double precision is 0.

A quick check to determine if the index is 0

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

>0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.64 void binned_dmnegate (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Negate manually specified binned double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.65 `void binned_dmpri (const int fold, const double * priX, const int incpriX, const double * carX, const int inccarX)`

Print manually specified binned double precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.66 `void binned_dmrenorm (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Renormalize manually specified binned double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.67 void binned_dmsetzero (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Set manually specified binned double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.68 double binned_dscale (const double *X*)

Get a reproducible double precision scale.

For any given *X*, return a reproducible scaling factor *Y* of the form

$2^{(\text{DBWIDTH} * z)}$ where *z* is an integer

such that

$Y * 2^{(-\text{DBL_MANT_DIG} - \text{DBWIDTH} - 1)} < X < Y * 2^{(\text{DBWIDTH} + 2)}$

Parameters

<i>X</i>	double precision number to be scaled
----------	--------------------------------------

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.69 float_binned* binned_sballot (const int *fold*)

binned single precision allocation

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.70 float binned_sbbound (const int *fold*, const int *N*, const float *X*, const float *S*)

Get binned single precision summation error bound.

This is a bound on the absolute error of a summation using binned types

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	the number of single precision floating point summands
<i>X</i>	the summand of maximum absolute value
<i>S</i>	the value of the sum computed using binned types

Returns

error bound

Author

Peter Ahrens

Date

31 Jul 2015

Author

Peter Ahrens
Hong Diep Nguyen

Date

21 May 2015

2.1.4.71 void binned_sbnegate (const int *fold*, float_binned * *X*)

Negate binned single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.72 int binned_sbnum (const int *fold*)

binned single precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in `float`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.73 `void binned_sbprint (const int fold, const float_binned * X)`

Print binned single precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.74 `void binned_sbrenorm (const int fold, float_binned * X)`

Renormalize binned single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.75 void binned_sbsadd (const int *fold*, const float *X*, float_binned * *Y*)Add single precision to binned single precision ($Y += X$)Performs the operation $Y += X$ on an binned type *Y*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.76 void binned_sbsbadd (const int *fold*, const float_binned * *X*, float_binned * *Y*)Add binned single precision ($Y += X$)Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.77 `float binned_sbsbaddsq (const int fold, const float scaleX, const float_binned * X, const float scaleY, float_binned * Y)`

Add binned single precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the binned types
<i>scaleX</i>	scale of X ($scaleX == \text{binned_sscale}(Z)$ for some <code>float Z</code>)
<i>X</i>	binned scalar X
<i>scaleY</i>	scale of Y ($scaleY == \text{binned_sscale}(Z)$ for some <code>float Z</code>)
<i>Y</i>	binned scalar Y

Returns

updated scale of Y

Author

Peter Ahrens

Date

2 Dec 2015

2.1.4.78 `void binned_sbsbaddv (const int fold, const int N, const float_binned * X, const int incX, float_binned * Y, const int incY)`

Add binned single precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	binned vector X
<i>incX</i>	X vector stride (use every $incX$ 'th element)
<i>Y</i>	binned vector Y
<i>incY</i>	Y vector stride (use every $incY$ 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.79 void binned_sbsbset (const int *fold*, const float_binned * *X*, float_binned * *Y*)Set binned single precision ($Y = X$)Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.80 size_t binned_sbsbze (const int *fold*)

binned single precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.81 void binned_sbsconv (const int *fold*, const float *X*, float_binned * *Y*)Convert single precision to binned single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.82 void binned_sbsdeposit (const int *fold*, const float *X*, float_binned * *Y*)

Add single precision to suitably binned binned single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [binned_sbsupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [binned_sbsdeposit\(\)](#) to deposit a maximum of [binned_SBENDURANCE](#) elements into *Y* before renormalizing *Y* with [binned_sbrenorm\(\)](#). After any number of successive calls of [binned_sbsdeposit\(\)](#) on *Y*, you must renormalize *Y* with [binned_sbrenorm\(\)](#) before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.83 void binned_sbsetzero (const int *fold*, float_binned * *X*)

Set binned single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.84 void `binned_sbsupdate` (const int *fold*, const float *X*, float_binned * *Y*)

Update binned single precision with single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.85 int `binned_index` (const float *X*)

Get index of single precision.

The index of a non-binned type is the smallest index an binned type would need to have to sum it reproducibly. Higher indices correspond to smaller bins.

Parameters

<i>X</i>	scalar X
----------	----------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.86 `const float* binned_smbins (const int X)`

Get binned single precision reference bins.

returns a pointer to the bins corresponding to the given index

Parameters

<i>X</i>	index
----------	-------

Returns

pointer to constant single precision bins of index *X*

Author

Peter Ahrens
Hong Diep Nguyen

Date

19 Jun 2015

2.1.4.87 `int binned_smdenorm (const int fold, const float * priX)`

Check if binned type has denormal bits.

A quick check to determine if calculations involving *X* cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the binned type
<i>priX</i>	<i>X</i> 's primary vector

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.88 int binned_sminindex (const float * *priX*)

Get index of manually specified binned single precision.

The index of an binned type is the bin that it corresponds to. Higher indicies correspond to smaller bins.

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

X's index

Author

Peter Ahrens
Hong Diep Nguyen

Date

23 Sep 2015

2.1.4.89 int binned_sminindex0 (const float * *priX*)

Check if index of manually specified binned single precision is 0.

A quick check to determine if the index is 0

Parameters

<i>priX</i>	X's primary vector
-------------	--------------------

Returns

>0 if x has index 0, 0 otherwise.

Author

Peter Ahrens

Date

19 May 2015

2.1.4.90 void `binned_smnegate` (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Negate manually specified binned single precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.91 void `binned_smprint` (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*)

Print manually specified binned single precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.92 void `binned_smrenorm` (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Renormalize manually specified binned single precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

23 Sep 2015

2.1.4.93 void `binned_smsadd` (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add single precision to manually specified binned single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.94 `void binned_smsconv (const int fold, const float X, float * priY, const int incpriY, float * carY, const int inccarY)`

Convert single precision to manually specified binned single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	<i>Y</i> 's carry vector
<i>inccarY</i>	stride within <i>Y</i> 's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

30 Apr 2015

2.1.4.95 `void binned_smsdeposit (const int fold, const float X, float * priY, const int incpriY)`

Add single precision to suitably binned manually specified binned single precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called [binned_smsupdate\(\)](#) on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call [binned_smsdeposit\(\)](#) to deposit a maximum of [binned_SBENDURANCE](#) elements into *Y* before renormalizing *Y* with [binned_smrenorm\(\)](#). After any number of successive calls of [binned_smsdeposit\(\)](#) on *Y*, you must renormalize *Y* with [binned_smrenorm\(\)](#) before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.96 void `binned_smsetzero` (const int *fold*, float * *priX*, const int *incpriX*, float * *carX*, const int *inccarX*)

Set manually specified binned single precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.97 void `binned_smsmadd` (const int *fold*, const float * *priX*, const int *incpriX*, const float * *carX*, const int *inccarX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add manually specified binned single precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.98 `float binned_smsmaddsq (const int fold, const float scaleX, const float * priX, const int incpriX, const float * carX, const int inccarX, const float scaleY, float * priY, const int incpriY, float * carY, const int inccarY)`

Add manually specified binned single precision scaled sums of squares ($Y += X$)

Performs the operation $Y += X$, where X and Y represent scaled sums of squares.

Parameters

<i>fold</i>	the fold of the binned types
<i>scaleX</i>	scale of X ($\text{scaleX} == \text{binned_sscale}(Z)$ for some <code>float Z</code>)
<i>priX</i>	X 's primary vector
<i>incpriX</i>	stride within X 's primary vector (use every incpriX 'th element)
<i>carX</i>	X 's carry vector
<i>inccarX</i>	stride within X 's carry vector (use every inccarX 'th element)
<i>scaleY</i>	scale of Y ($\text{scaleY} == \text{binned_sscale}(Z)$ for some <code>double Z</code>)
<i>priY</i>	Y 's primary vector
<i>incpriY</i>	stride within Y 's primary vector (use every incpriY 'th element)
<i>carY</i>	Y 's carry vector
<i>inccarY</i>	stride within Y 's carry vector (use every inccarY 'th element)

Returns

updated scale of Y

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.99 `void binned_smsmset (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX, float * priY, const int incpriY, float * carY, const int inccarY)`

Set manually specified binned single precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.100 void `binned_smsrescale` (const int *fold*, const float *X*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

rescale manually specified binned single precision sum of squares

Rescale an binned single precision sum of squares Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	Y's new scaleY ($X == \text{binned_sscale}(f)$ for some <code>float f</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{binned_sscale}(f)$ for some <code>float f</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.101 void `binmed_smsupdate` (const int *fold*, const float *X*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Update manually specified binned single precision with single precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

5 May 2015

2.1.4.102 float binned_ssbconv (const int *fold*, const float_binned * *X*)

Convert binned single precision to single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.103 float binned_sscalescale (const float *X*)

Get a reproducible single precision scale.

For any given X, return a reproducible scaling factor Y of the form

$2^{(\text{SBWIDTH} * z)}$ where z is an integer

such that

$Y * 2^{(-\text{FLT_MANT_DIG} - \text{SBWIDTH} - 1)} < X < Y * 2^{(\text{SBWIDTH} + 2)}$

Parameters

<i>X</i>	single precision number to be scaled
----------	--------------------------------------

Returns

reproducible scaling factor

Author

Peter Ahrens

Date

19 Jun 2015

2.1.4.104 `float binned_ssmconv (const int fold, const float * priX, const int incpriX, const float * carX, const int inccarX)`

Convert manually specified binned single precision to single precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Returns

scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.105 `double binned_ufp (double X)`

unit in the first place

This method returns just the implicit 1 in the mantissa of a `double`

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.106 float binned_ufpf (float *X*)

unit in the first place

This method returns just the implicit 1 in the mantissa of a `float`

Parameters

<i>X</i>	scalar <i>X</i>
----------	-----------------

Returns

unit in the first place

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.107 double_complex_binned* binned_zballoc (const int *fold*)

binned complex double precision allocation

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

a freshly allocated binned type. (free with `free()`)

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.108 `void binned_zbdbset (const int fold, const double_binned * X, double_complex_binned * Y)`

Set binned complex double precision to binned double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.109 `void binned_zbduupdate (const int fold, const double X, double_complex_binned * Y)`

Update binned complex double precision with double precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value less than *X*

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.110 void `binned_zbnegate` (const int *fold*, `double_complex_binned` * *X*)

Negate binned complex double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.111 int `binned_zbnum` (const int *fold*)

binned complex double precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in `double`) of the binned type

Author

Peter Ahrens

Date

27 Apr 2015

2.1.4.112 void binned_zbprint (const int *fold*, const double_complex_binned * *X*)

Print binned complex double precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.113 void binned_zbrenorm (const int *fold*, double_complex_binned * *X*)

Renormalize binned complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.114 void binned_zbsetzero (const int *fold*, double_complex_binned * *X*)

Set binned double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.115 `size_t binned_zbsize (const int fold)`

binned complex double precision size

Parameters

<i>fold</i>	the fold of the binned type
-------------	-----------------------------

Returns

the size (in bytes) of the binned type

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.116 `void binned_zbadd (const int fold, const void * X, double_complex_binned * Y)`

Add complex double precision to binned complex double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.117 `void binned_zbzbadd (const int fold, const double_complex_binned * X, double_complex_binned * Y)`

Add binned complex double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.118 `void binned_zbzbaddv (const int fold, const int N, const double_complex_binned * X, const int incX, double_complex_binned * Y, const int incY)`

Add binned complex double precision vectors ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	binned vector <i>X</i>
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned vector <i>Y</i>
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

25 Jun 2015

2.1.4.119 void binned_zbzbset (const int *fold*, const double_complex_binned * *X*, double_complex_binned * *Y*)

Set binned complex double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.120 void binned_zbzconv (const int *fold*, const void * *X*, double_complex_binned * *Y*)

Convert complex double precision to binned complex double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>Y</i>	binned scalar <i>Y</i>

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.121 void binned_zbzdeposit (const int *fold*, const void * *X*, double_complex_binned * *Y*)

Add complex double precision to suitably binned binned complex double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called `binned_zbupdate()` on Y with the maximum absolute value of all future elements you wish to deposit in Y, you can call `binned_zbdeposit()` to deposit a maximum of `binned_DBENDURANCE` elements into Y before renormalizing Y with `binned_zbrenorm()`. After any number of successive calls of `binned_zbdeposit()` on Y, you must renormalize Y with `binned_zbrenorm()` before using any other function on Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.122 void `binned_zbupdate (const int fold, const void * X, double_complex_binned * Y)`

Update binned complex double precision with complex double precision ($X \rightarrow Y$)

This method updates *Y* to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of *X* respectively.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>Y</i>	binned scalar Y

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.123 int `binned_zmdenorm (const int fold, const double * priX)`

Check if binned type has denormal bits.

A quick check to determine if calculations involving *X* cannot be performed with "denormals are zero"

Parameters

<i>fold</i>	the fold of the binned type
<i>priX</i>	<i>X</i> 's primary vector

Returns

>0 if x has denormal bits, 0 otherwise.

Author

Peter Ahrens

Date

23 Jun 2015

2.1.4.124 void `binned_zmdmset` (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Set manually specified binned complex double precision to manually specified binned double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.125 void `binned_zmdrescale` (const int *fold*, const double *X*, const double *scaleY*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

rescale manually specified binned complex double precision sum of squares

Rescale an binned complex double precision sum of squares Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	Y's new scaleY ($X == \text{binned_dscale}(f)$ for some <code>double f</code>) ($X \geq \text{scaleY}$)
<i>scaleY</i>	Y's current scaleY ($\text{scaleY} == \text{binned_dscale}(f)$ for some <code>double f</code>) ($X \geq \text{scaleY}$)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

1 Jun 2015

2.1.4.126 `void binned_zmdupdate (const int fold, const double X, double * priY, const int incpriY, double * carY, const int inccarY)`

Update manually specified binned complex double precision with double precision ($X \rightarrow Y$)

This method updates Y to an index suitable for adding numbers with absolute value less than X

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.127 `void binned_zmnegate (const int fold, double * priX, const int incpriX, double * carX, const int inccarX)`

Negate manually specified binned complex double precision ($X = -X$)

Performs the operation $X = -X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.128 void binned_zmprint (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*)

Print manually specified binned complex double precision.

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.129 void binned_zmrenorm (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Renormalize manually specified binned complex double precision.

Renormalization keeps the primary vector within the necessary bins by shifting over to the carry vector

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.130 void `binned_zmsetzero` (const int *fold*, double * *priX*, const int *incpriX*, double * *carX*, const int *inccarX*)

Set manually specified binned complex double precision to 0 ($X = 0$)

Performs the operation $X = 0$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.131 void `binned_zmzadd` (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add complex double precision to manually specified binned complex double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type Y

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.132 void `binned_zmzconv` (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Convert complex double precision to manually specified binned complex double precision ($X \rightarrow Y$)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	<i>Y</i> 's carry vector
<i>inccarY</i>	stride within <i>Y</i> 's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.133 void `binned_zmzdeposit` (const int *fold*, const void * *X*, double * *priY*, const int *incpriY*)

Add complex double precision to suitably binned manually specified binned complex double precision ($Y += X$)

Performs the operation $Y += X$ on an binned type *Y* where the index of *Y* is larger than the index of *X*

Note

This routine was provided as a means of allowing the you to optimize your code. After you have called `binned_zmzupdate()` on *Y* with the maximum absolute value of all future elements you wish to deposit in *Y*, you can call `binned_zmzdeposit()` to deposit a maximum of `binned_DBENDURANCE` elements into *Y* before renormalizing *Y* with `binned_zmrenorm()`. After any number of successive calls of `binned_zmzdeposit()` on *Y*, you must renormalize *Y* with `binned_zmrenorm()` before using any other function on *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar <i>X</i>
<i>priY</i>	<i>Y</i> 's primary vector
<i>incpriY</i>	stride within <i>Y</i> 's primary vector (use every <i>incpriY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

10 Jun 2015

2.1.4.134 void `binned_zmzmadd` (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add manually specified binned complex double precision ($Y += X$)

Performs the operation $Y += X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every <i>inccarX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.135 void `binned_zmzmset` (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Set manually specified binned complex double precision ($Y = X$)

Performs the operation $Y = X$

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every <i>incpriX</i> 'th element)

Parameters

<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.136 `void binned_zmzupdate (const int fold, const void * X, double * priY, const int incpriY, double * carY, const int inccarY)`

Update manually specified binned complex double precision with complex double precision (X -> Y)

This method updates Y to an index suitable for adding numbers with absolute value of real and imaginary components less than absolute value of real and imaginary components of X respectively.

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	scalar X
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.137 `void binned_zzbconv_sub (const int fold, const double_complex_binned * X, void * conv)`

Convert binned complex double precision to complex double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the binned types
<i>X</i>	binned scalar X
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.1.4.138 void `binned_zzmconv_sub` (const int *fold*, const double * *priX*, const int *incpriX*, const double * *carX*, const int *inccarX*, void * *conv*)

Convert manually specified binned complex double precision to complex double precision (X -> Y)

Parameters

<i>fold</i>	the fold of the binned types
<i>priX</i>	X's primary vector
<i>incpriX</i>	stride within X's primary vector (use every incpriX'th element)
<i>carX</i>	X's carry vector
<i>inccarX</i>	stride within X's carry vector (use every inccarX'th element)
<i>conv</i>	scalar return

Author

Hong Diep Nguyen
Peter Ahrens

Date

27 Apr 2015

2.2 include/binnedBLAS.h File Reference

[binnedBLAS.h](#) defines BLAS Methods that operate on binned types.

```
#include "binned.h"
#include "reproBLAS.h"
```

Functions

- float [binnedBLAS_samax](#) (const int N, const float *X, const int incX)
Find maximum absolute value in vector of single precision.
- double [binnedBLAS_damax](#) (const int N, const double *X, const int incX)
Find maximum absolute value in vector of double precision.
- void [binnedBLAS_camax_sub](#) (const int N, const void *X, const int incX, void *amax)
Find maximum magnitude in vector of complex single precision.
- void [binnedBLAS_zamax_sub](#) (const int N, const void *X, const int incX, void *amax)
Find maximum magnitude in vector of complex double precision.
- float [binnedBLAS_samaxm](#) (const int N, const float *X, const int incX, const float *Y, const int incY)
Find maximum absolute value pairwise product between vectors of single precision.
- double [binnedBLAS_damaxm](#) (const int N, const double *X, const int incX, const double *Y, const int incY)
Find maximum absolute value pairwise product between vectors of double precision.
- void [binnedBLAS_camaxm_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)
Find maximum magnitude pairwise product between vectors of complex single precision.
- void [binnedBLAS_zamaxm_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *amaxm)
Find maximum magnitude pairwise product between vectors of complex double precision.
- void [binnedBLAS_dbdsum](#) (const int fold, const int N, const double *X, const int incX, [double_binned](#) *Y)
Add to binned double precision Y the sum of double precision vector X.
- void [binnedBLAS_dmdsum](#) (const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified binned double precision Y the sum of double precision vector X.
- void [binnedBLAS_dbdasum](#) (const int fold, const int N, const double *X, const int incX, [double_binned](#) *Y)
Add to binned double precision Y the absolute sum of double precision vector X.
- void [binnedBLAS_dmdasum](#) (const int fold, const int N, const double *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified binned double precision Y the absolute sum of double precision vector X.
- double [binnedBLAS_dbdssq](#) (const int fold, const int N, const double *X, const int incX, const double scaleY, [double_binned](#) *Y)
Add to scaled binned double precision Y the scaled sum of squares of elements of double precision vector X.
- double [binnedBLAS_dmdssq](#) (const int fold, const int N, const double *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of double precision vector X.
- void [binnedBLAS_dbddot](#) (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, [double_binned](#) *Z)
Add to binned double precision Z the dot product of double precision vectors X and Y.
- void [binnedBLAS_dmddot](#) (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)
Add to manually specified binned double precision Z the dot product of double precision vectors X and Y.
- void [binnedBLAS_zbzsum](#) (const int fold, const int N, const void *X, const int incX, [double_binned](#) *Y)
Add to binned complex double precision Y the sum of complex double precision vector X.
- void [binnedBLAS_zmzsum](#) (const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)
Add to manually specified binned complex double precision Y the sum of complex double precision vector X.
- void [binnedBLAS_dbzasum](#) (const int fold, const int N, const void *X, const int incX, [double_binned](#) *Y)
Add to binned double precision Y the absolute sum of complex double precision vector X.
- void [binnedBLAS_dmzasum](#) (const int fold, const int N, const void *X, const int incX, double *priY, const int incpriY, double *carY, const int inccarY)

- Add to manually specified binned double precision Y the absolute sum of complex double precision vector X.*
- double `binnedBLAS_dbzssq` (const int fold, const int N, const void *X, const int incX, const double scaleY, `double_binned` *Y)
- Add to scaled binned double precision Y the scaled sum of squares of elements of complex double precision vector X.*
- double `binnedBLAS_dmzssq` (const int fold, const int N, const void *X, const int incX, const double scaleY, double *priY, const int incpriY, double *carY, const int inccarY)
- Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of complex double precision vector X.*
- void `binnedBLAS_zbzdotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, `double_binned` *Z)
- Add to binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.*
- void `binnedBLAS_zmzdotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)
- Add to manually specified binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.*
- void `binnedBLAS_zbzdorc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, `double_binned` *Z)
- Add to binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.*
- void `binnedBLAS_zmzdorc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, double *manZ, const int incmanZ, double *carZ, const int inccarZ)
- Add to manually specified binned complex double precision Z the conjugated dot product of complex double precision vectors X and Y.*
- void `binnedBLAS_sbssum` (const int fold, const int N, const float *X, const int incX, `float_binned` *Y)
- Add to binned single precision Y the sum of single precision vector X.*
- void `binnedBLAS_smssum` (const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)
- Add to manually specified binned single precision Y the sum of single precision vector X.*
- void `binnedBLAS_sbsasum` (const int fold, const int N, const float *X, const int incX, `float_binned` *Y)
- Add to binned single precision Y the absolute sum of single precision vector X.*
- void `binnedBLAS_smsasum` (const int fold, const int N, const float *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)
- Add to manually specified binned single precision Y the absolute sum of double precision vector X.*
- float `binnedBLAS_sbsssq` (const int fold, const int N, const float *X, const int incX, const float scaleY, `float_binned` *Y)
- Add to scaled binned single precision Y the scaled sum of squares of elements of single precision vector X.*
- float `binnedBLAS_smsssq` (const int fold, const int N, const float *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)
- Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of single precision vector X.*
- void `binnedBLAS_sbsdorc` (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, `float_binned` *Z)
- Add to binned single precision Z the dot product of single precision vectors X and Y.*
- void `binnedBLAS_smsdorc` (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)
- Add to manually specified binned single precision Z the dot product of single precision vectors X and Y.*
- void `binnedBLAS_cbcsum` (const int fold, const int N, const void *X, const int incX, `float_binned` *Y)
- Add to binned complex single precision Y the sum of complex single precision vector X.*
- void `binnedBLAS_cmcsun` (const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)
- Add to manually specified binned complex single precision Y the sum of complex single precision vector X.*
- void `binnedBLAS_sbcasun` (const int fold, const int N, const void *X, const int incX, `float_binned` *Y)
- Add to binned single precision Y the absolute sum of complex single precision vector X.*

- void `binnedBLAS_smcasum` (const int fold, const int N, const void *X, const int incX, float *priY, const int incpriY, float *carY, const int inccarY)
Add to manually specified binned single precision Y the absolute sum of complex single precision vector X.
- float `binnedBLAS_sbcssq` (const int fold, const int N, const void *X, const int incX, const float scaleY, `float↵_binned` *Y)
Add to scaled binned single precision Y the scaled sum of squares of elements of complex single precision vector X.
- float `binnedBLAS_smcssq` (const int fold, const int N, const void *X, const int incX, const float scaleY, float *priY, const int incpriY, float *carY, const int inccarY)
Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of complex single precision vector X.
- void `binnedBLAS_cbcddotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, `float_binned` *Z)
Add to binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void `binnedBLAS_cmcdotu` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)
Add to manually specified binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.
- void `binnedBLAS_cbcddotc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, `float_binned` *Z)
Add to binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void `binnedBLAS_cmcdotc` (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, float *manZ, const int incmanZ, float *carZ, const int inccarZ)
Add to manually specified binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.
- void `binnedBLAS_dbdgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, `double_binned` *Y, const int incY)
Add to binned double precision vector Y the matrix-vector product of double precision matrix A and double precision vector X.
- void `binnedBLAS_dbdgemm` (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, `double_binned` *C, const int ldc)
Add to binned double precision matrix C the matrix-matrix product of double precision matrices A and B.
- void `binnedBLAS_sbsgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, `float_binned` *Y, const int incY)
Add to binned single precision vector Y the matrix-vector product of single precision matrix A and single precision vector X.
- void `binnedBLAS_sbsgemm` (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, `float_binned` *C, const int ldc)
Add to binned single precision matrix C the matrix-matrix product of single precision matrices A and B.
- void `binnedBLAS_zbzgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, `double_complex_binned` *Y, const int incY)
Add to binned complex double precision vector Y the matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void `binnedBLAS_zbzgemm` (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, `double_complex_binned` *C, const int ldc)
Add to binned complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.
- void `binnedBLAS_cbcgemv` (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, `float_complex_binned` *Y, const int incY)

Add to binned complex single precision vector Y the matrix-vector product of complex single precision matrix A and complex single precision vector X.

- void [binnedBLAS_cbcgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, [float_complex_binned](#) *C, const int ldc)

Add to binned complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.

2.2.1 Detailed Description

[binnedBLAS.h](#) defines BLAS Methods that operate on binned types.

This header is modeled after cblas.h, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `df00` would perform the function `f00` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double ([double_binned](#))
- zb - binned complex double ([double_complex_binned](#))
- sb - binned float ([float_binned](#))
- cb - binned complex float ([float_complex_binned](#))
- dm - manually specified binned double (`double, double`)
- zm - manually specified binned complex double (`double, double`)
- sm - manually specified binned float (`float, float`)
- cm - manually specified binned complex float (`float, float`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in `config.h`. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use. Power users of the library may find themselves wanting to manually specify the underlying primary and carry vectors of an binned type themselves. If you do not know what these are, don't worry about the manually specified binned types.

2.2.2 Function Documentation

2.2.2.1 void binnedBLAS_camax_sub (const int *N*, const void * *X*, const int *incX*, void * *amax*)

Find maximum magnitude in vector of complex single precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>amax</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.2 void binnedBLAS_camaxm_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *amaxm*)

Find maximum magnitude pairwise product between vectors of complex single precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.3 void binnedBLAS_cbcddotc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_binned * *Z*)

Add to binned complex single precision *Z* the conjugated dot product of complex single precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and conjugated *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	binned scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.4 void binnedBLAS_cbcddotu (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float_complex_binned * *Z*)

Add to binned complex single precision *Z* the unconjugated dot product of complex single precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	binned scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.5 void binnedBLAS_cbcgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, float_complex_binned * *C*, const int *ldc*)

Add to binned complex single precision matrix C the matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where op(X) is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	binned complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.6 void binnedBLAS_cbcgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, float_complex_binned * *Y*, const int *incY*)

Add to binned complex single precision vector *Y* the matrix-vector product of complex single precision matrix *A* and complex single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$ or $y := \alpha * A^T * x + y$ or $y := \alpha * A^H * x + y$,

where α is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar α
<i>A</i>	complex single precision matrix of dimension (<i>M</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>N</i>) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	complex single precision vector of at least size <i>N</i> if not transposed or size <i>M</i> otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned complex single precision vector <i>Y</i> of at least size <i>M</i> if not transposed or size <i>N</i> otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.7 void binnedBLAS_cbcsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_complex_binned * *Y*)

Add to binned complex single precision *Y* the sum of complex single precision vector *X*.

Add to *Y* the binned sum of *X*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar <i>Y</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.8 void `binnedBLAS_cmcdotc` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float * *priZ*, const int *incpriZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified binned complex single precision Z the conjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and conjugated Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.9 void `binnedBLAS_cmcdotu` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, float * *priZ*, const int *incpriZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified binned complex single precision Z the unconjugated dot product of complex single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length

Parameters

<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.10 void binnedBLAS_cmcsun (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned complex single precision Y the sum of complex single precision vector X.

Add to Y the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.11 double binnedBLAS_damax (const int *N*, const double * *X*, const int *incX*)

Find maximum absolute value in vector of double precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute maximum value of *X*

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.12 double binnedBLAS_damaxm (const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*)

Find maximum absolute value pairwise product between vectors of double precision.

Returns the absolute value of the pairwise product of maximum absolute value between *X* and *Y*.

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

absolute maximum value multiple of *X* and *Y*

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.13 void binBLAS_dbdasum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision *Y* the absolute sum of double precision vector *X*.

Add to *Y* the binned sum of absolute values of elements in *X*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar <i>Y</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.14 void binBLAS_dbddot (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*, double_binned * *Z*)

Add to binned double precision *Z* the dot product of double precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	binned scalar <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.15 void binnedBLAS_dbdgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, double_binned * *C*, const int *ldc*)

Add to binned double precision matrix C the matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α is a scalar, A and B are matrices with $\text{op}(A)$ an M by K matrix and $\text{op}(B)$ a K by N matrix, and C is an binned M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	binned double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.16 void binBLAS_dbdgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double * *A*, const int *lda*, const double * *X*, const int *incX*, double_binned * *Y*, const int *incY*)

Add to binned double precision vector *Y* the matrix-vector product of double precision matrix *A* and double precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$ or $y := \alpha * A^T * x + y$,

where *alpha* is a scalar, *x* is a vector, *y* is an binned vector, and *A* is an *M* by *N* matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar <i>alpha</i>
<i>A</i>	double precision matrix of dimension (<i>M</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>N</i>) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	double precision vector of at least size <i>N</i> if not transposed or size <i>M</i> otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned double precision vector <i>Y</i> of at least size <i>M</i> if not transposed or size <i>N</i> otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.17 double binBLAS_dbdssq (const int *fold*, const int *N*, const double * *X*, const int *incX*, const double *scaleY*, double_binned * *Y*)

Add to scaled binned double precision *Y* the scaled sum of squares of elements of double precision vector *X*.

Add to *Y* the scaled binned sum of the squares of each element of *X*. The scaling of each square is performed using [binned_dscaled\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of <i>Y</i>
<i>Y</i>	binned scalar <i>Y</i>

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.18 void binnedBLAS_dbdsum (const int *fold*, const int *N*, const double * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision Y the sum of double precision vector X.

Add to Y the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.19 void binnedBLAS_dbzasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_binned * *Y*)

Add to binned double precision Y the absolute sum of complex double precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.20 `double binnedBLAS_dbzssq (const int fold, const int N, const void * X, const int incX, const double scaleY, double_binned * Y)`

Add to scaled binned double precision *Y* the scaled sum of squares of elements of complex double precision vector *X*.

Add to *Y* the scaled binned sum of the squares of each element of *X*. The scaling of each square is performed using [binned_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of <i>Y</i>
<i>Y</i>	binned scalar <i>Y</i>

Returns

the new scaling factor of *Y*

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.21 `void binnedBLAS_dmdasum (const int fold, const int N, const double * X, const int incX, double * priY, const int incpriY, double * carY, const int inccarY)`

Add to manually specified binned double precision *Y* the absolute sum of double precision vector *X*.

Add to *Y* the binned sum of absolute values of elements in *X*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length

Parameters

<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.22 `void binnedBLAS_dmdot (const int fold, const int N, const double * X, const int incX, const double * Y, const int incY, double * priZ, const int incpriZ, double * carZ, const int inccarZ)`

Add to manually specified binned double precision Z the dot product of double precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.23 `double binnedBLAS_dmdssq (const int fold, const int N, const double * X, const int incX, const double scaleY, double * priY, const int incpriY, double * carY, const int inccarY)`

Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using [binned_dscale\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.24 `void binnedBLAS_dmdsum (const int fold, const int N, const double * X, const int incX, double * priY, const int incpriY, double * carY, const int inccarY)`

Add to manually specified binned double precision Y the sum of double precision vector X.

Set Y to the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.25 void binnedBLAS_dmzasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add to manually specified binned double precision Y the absolute sum of complex double precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.26 double binnedBLAS_dmzssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const double *scaleY*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add to scaled manually specified binned double precision Y the scaled sum of squares of elements of complex double precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using [binned_dscalet\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)

Parameters

<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.27 float binnedBLAS_samax (const int *N*, const float * *X*, const int *incX*)

Find maximum absolute value in vector of single precision.

Returns the absolute value of the element of maximum absolute value in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)

Returns

absolute maximum value of X

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.28 float binnedBLAS_samaxm (const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Find maximum absolute value pairwise product between vectors of single precision.

Returns the absolute value of the pairwise product of maximum absolute value between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

absolute maximum value multiple of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.29 void `binnedBLAS_sbcasum` (const int *fold*, const int *N*, const void * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the absolute sum of complex single precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.30 float `binnedBLAS_sbcssq` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const float *scaleY*, float_binned * *Y*)

Add to scaled binned single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using [binned_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.31 void binnedBLAS_sbsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the absolute sum of single precision vector X.

Add to Y the binned sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.32 void binnedBLAS_sbsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float_binned * *Z*)

Add to binned single precision Z the dot product of single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every incY'th element)
<i>Z</i>	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.33 void binnedBLAS_sbsgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, float_binned * *C*, const int *ldc*)

Add to binned single precision matrix C the matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C$,

where op(X) is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

alpha is a scalar, A and B are matrices with op(A) an M by K matrix and op(B) a K by N matrix, and C is an binned M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>C</i>	binned single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.34 void binnedBLAS_sbsgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, float_binned * *Y*, const int *incY*)

Add to binned single precision vector *Y* the matrix-vector product of single precision matrix *A* and single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$ or $y := \alpha * A^T * x + y$,

where α is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar α
<i>A</i>	single precision matrix of dimension (<i>M</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>N</i>) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	single precision vector of at least size <i>N</i> if not transposed or size <i>M</i> otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned single precision vector <i>Y</i> of at least size <i>M</i> if not transposed or size <i>N</i> otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.35 float binnedBLAS_sbsssq (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float *scaleY*, float_binned * *Y*)

Add to scaled binned single precision *Y* the scaled sum of squares of elements of single precision vector *X*.

Add to *Y* the scaled binned sum of the squares of each element of *X*. The scaling of each square is performed using [binned_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>Y</i>	binned scalar Y

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.36 void binnedBLAS_sbssum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float_binned * *Y*)

Add to binned single precision Y the sum of single precision vector X.

Add to Y the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	binned scalar Y

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.37 void binnedBLAS_smcasum (const int *fold*, const int *N*, const void * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned single precision Y the absolute sum of complex single precision vector X.

Add to Y the binned sum of magnitudes of elements of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.38 float binnedBLAS_smcssq (const int *fold*, const int *N*, const void * *X*, const int *incX*, const float *scaleY*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of complex single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using [binned_sscales\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.39 void binnedBLAS_smsasum (const int *fold*, const int *N*, const float * *X*, const int *incX*, float * *priY*, const int *incpriY*, float * *carY*, const int *inccarY*)

Add to manually specified binned single precision Y the absolute sum of double precision vector X.

Add to Y to the binned sum of absolute values of elements in X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.40 void binnedBLAS_smsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*, float * *priZ*, const int *incpriZ*, float * *carZ*, const int *inccarZ*)

Add to manually specified binned single precision Z the dot product of single precision vectors X and Y.

Add to Z the binned sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.41 `float binnedBLAS_smsssq (const int fold, const int N, const float * X, const int incX, const float scaleY, float * priY, const int incpriY, float * carY, const int inccarY)`

Add to scaled manually specified binned single precision Y the scaled sum of squares of elements of single precision vector X.

Add to Y the scaled binned sum of the squares of each element of X. The scaling of each square is performed using [binned_sscale\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>scaleY</i>	the scaling factor of Y
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every <i>incpriY</i> 'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every <i>inccarY</i> 'th element)

Returns

the new scaling factor of Y

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.42 `void binnedBLAS_smssum (const int fold, const int N, const float * X, const int incX, float * priY, const int incpriY, float * carY, const int inccarY)`

Add to manually specified binned single precision Y the sum of single precision vector X.

Add to Y the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.43 `void binnedBLAS_zamax_sub (const int N, const void * X, const int incX, void * amax)`

Find maximum magnitude in vector of complex double precision.

Returns the magnitude of the element of maximum magnitude in an array.

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>amax</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.44 `void binnedBLAS_zamaxm_sub (const int N, const void * X, const int incX, const void * Y, const int incY, void * amaxm)`

Find maximum magnitude pairwise product between vectors of complex double precision.

Returns the magnitude of the pairwise product of maximum magnitude between X and Y.

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>amaxm</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.45 void `binnedBLAS_zbzdetc` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_binned * *Z*)

Add to binned complex double precision *Z* the conjugated dot product of complex double precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and conjugated *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>Z</i>	scalar return <i>Z</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.46 void `binnedBLAS_zbzdotu` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double_complex_binned * *Z*)

Add to binned complex double precision *Z* the unconjugated dot product of complex double precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every incY'th element)
<i>Z</i>	binned scalar Z

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.47 `void binnedBLAS_zbzgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void * alpha, const void * A, const int lda, const void * B, const int ldb, double_complex_binned * C, const int ldc)`

Add to binned complex double precision matrix C the matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + C,$

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$ or $\text{op}(X) = X^{**H},$

α is a scalar, A and B are matrices with $\text{op}(A)$ an M by K matrix and $\text{op}(B)$ a K by N matrix, and C is an binned M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.

Parameters

<i>lda</i>	the first dimension of A as declared in the calling program. <i>lda</i> must be at least <i>na</i> in row major or <i>ma</i> in column major.
<i>B</i>	complex double precision matrix of dimension (<i>mb</i> , <i>ldb</i>) in row-major or (<i>ldb</i> , <i>nb</i>) in column-major. (<i>mb</i> , <i>nb</i>) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. <i>ldb</i> must be at least <i>nb</i> in row major or <i>mb</i> in column major.
<i>C</i>	binned complex double precision matrix of dimension (M, <i>ldc</i>) in row-major or (<i>ldc</i> , N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. <i>ldc</i> must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.48 void `binnedBLAS_zbgemv` (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, double_complex_binned * *Y*, const int *incY*)

Add to binned complex double precision vector *Y* the matrix-vector product of complex double precision matrix *A* and complex double precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + y$ or $y := \alpha * A^T * x + y$ or $y := \alpha * A^H * x + y$,

where α is a scalar, x is a vector, y is an binned vector, and A is an M by N matrix.

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, <i>lda</i>) in row-major or (<i>lda</i> , N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.2.2.49 void binnedBLAS_zbzsum (const int *fold*, const int *N*, const void * *X*, const int *incX*, double_complex_binned * *Y*)

Add to binned complex double precision *Y* the sum of complex double precision vector *X*.

Add to *Y* the binned sum of *X*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	binned scalar <i>Y</i>

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.50 void binnedBLAS_zmzdetc (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double * *priZ*, const int *incpriZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified binned complex double precision *Z* the conjugated dot product of complex double precision vectors *X* and *Y*.

Add to *Z* the binned sum of the pairwise products of *X* and conjugated *Y*.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	<i>Z</i> 's primary vector
<i>incpriZ</i>	stride within <i>Z</i> 's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	<i>Z</i> 's carry vector
<i>inccarZ</i>	stride within <i>Z</i> 's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.51 void `binnedBLAS_zmzdotu` (const int *fold*, const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, double * *priZ*, const int *incpriZ*, double * *carZ*, const int *inccarZ*)

Add to manually specified binned complex double precision Z the unconjugated dot product of complex double precision vectors X and Y.

Add to Z to the binned sum of the pairwise products of X and Y.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>priZ</i>	Z's primary vector
<i>incpriZ</i>	stride within Z's primary vector (use every <i>incpriZ</i> 'th element)
<i>carZ</i>	Z's carry vector
<i>inccarZ</i>	stride within Z's carry vector (use every <i>inccarZ</i> 'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.2.2.52 void `binnedBLAS_zmzsum` (const int *fold*, const int *N*, const void * *X*, const int *incX*, double * *priY*, const int *incpriY*, double * *carY*, const int *inccarY*)

Add to manually specified binned complex double precision Y the sum of complex double precision vector X.

Add to Y the binned sum of X.

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector

Parameters

<i>incX</i>	X vector stride (use every incX'th element)
<i>priY</i>	Y's primary vector
<i>incpriY</i>	stride within Y's primary vector (use every incpriY'th element)
<i>carY</i>	Y's carry vector
<i>inccarY</i>	stride within Y's carry vector (use every inccarY'th element)

Author

Peter Ahrens

Date

15 Jan 2016

2.3 include/binnedMPI.h File Reference

[binnedMPI.h](#) defines MPI wrapper functions for binned types and the necessary functions to perform reproducible reductions.

```
#include <mpi.h>
#include "binned.h"
```

Functions

- MPI_Op [binnedMPI_DBDBADD](#) (const int fold)
Get an MPI_OP to add binned double precision ($Y += X$)
- MPI_Op [binnedMPI_ZBZBADD](#) (const int fold)
Get an MPI_OP to add binned complex double precision ($Y += X$)
- MPI_Op [binnedMPI_SBSBADD](#) (const int fold)
Get an MPI_OP to add binned double precision ($Y += X$)
- MPI_Op [binnedMPI_CBCBADD](#) (const int fold)
Get an MPI_OP to add binned complex single precision ($Y += X$)
- MPI_Op [binnedMPI_DBDBADDSQ](#) (const int fold)
Get an MPI_OP to add binned double precision scaled sums of squares ($Y += X$)
- MPI_Op [binnedMPI_SBSBADDSQ](#) (const int fold)
Get an MPI_OP to add binned single precision scaled sums of squares ($Y += X$)
- MPI_Datatype [binnedMPI_DOUBLE_BINNED](#) (const int fold)
Get an MPI_DATATYPE representing binned double precision.
- MPI_Datatype [binnedMPI_DOUBLE_COMPLEX_BINNED](#) (const int fold)
Get an MPI_DATATYPE representing binned complex double precision.
- MPI_Datatype [binnedMPI_FLOAT_BINNED](#) (const int fold)
Get an MPI_DATATYPE representing binned single precision.
- MPI_Datatype [binnedMPI_FLOAT_COMPLEX_BINNED](#) (const int fold)
Get an MPI_DATATYPE representing binned complex single precision.
- MPI_Datatype [binnedMPI_DOUBLE_BINNED_SCALED](#) (const int fold)
Get an MPI_DATATYPE representing scaled binned double precision.
- MPI_Datatype [binnedMPI_FLOAT_BINNED_SCALED](#) (const int fold)
Get an MPI_DATATYPE representing scaled binned single precision.

2.3.1 Detailed Description

[binnedMPI.h](#) defines MPI wrapper functions for binned types and the necessary functions to perform reproducible reductions.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- d - double (`double`)
- z - complex double (`*void`)
- s - float (`float`)
- c - complex float (`*void`)
- db - binned double ([double_binned](#))
- zb - binned complex double ([double_complex_binned](#))
- sb - binned float ([float_binned](#))
- cb - binned complex float ([float_complex_binned](#))

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in `config.h`. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.

2.3.2 Function Documentation

2.3.2.1 MPI_Op binnedMPI_CBCBADD (const int *fold*)

Get an `MPI_OP` to add binned complex single precision ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ on two arrays of binned complex single precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_FLOAT_COMPLEX_BINNED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.2 MPI_Op binnedMPI_DBDBADD (const int *fold*)

Get an MPI_OP to add binned double precision ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ on two arrays of binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_DOUBLE_BINNED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.3 MPI_Op binnedMPI_DBDBADDSQ (const int *fold*)

Get an MPI_OP to add binned double precision scaled sums of squares ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ where X and Y represent scaled sums of squares on two arrays of scaled binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_DOUBLE_BINNED_SCALED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.4 MPI_Datatype binnedMPI_DOUBLE_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents an binned double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.5 MPI_Datatype binnedMPI_DOUBLE_BINNED_SCALED (const int *fold*)

Get an MPI_DATATYPE representing scaled binned double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a scaled binned double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.6 MPI_Datatype binnedMPI_DOUBLE_COMPLEX_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned complex double precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents an binned complex double precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.7 MPI_Datatype binnedMPI_FLOAT_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a binned single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.8 MPI_Datatype binnedMPI_FLOAT_BINNED_SCALED (const int *fold*)

Get an MPI_DATATYPE representing scaled binned single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a scaled binned single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.9 MPI_Datatype binnedMPI_FLOAT_COMPLEX_BINNED (const int *fold*)

Get an MPI_DATATYPE representing binned complex single precision.

Creates (if it has not already been created) and returns a datatype handle for an MPI datatype that represents a binned complex single precision type.

This method may call `MPI_Type_contiguous()` and `MPI_Type_commit()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.10 MPI_Op binnedMPI_SBSBADD (const int *fold*)

Get an MPI_OP to add binned double precision ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ on two arrays of binned double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_FLOAT_BINNED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.11 MPI_Op binnedMPI_SBSBADDQ (const int *fold*)

Get an MPI_OP to add binned single precision scaled sums of squares ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ where X and Y represent scaled sums of squares on two arrays of scaled binned single precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_FLOAT_BINNED_SCALED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.3.2.12 MPI_Op binnedMPI_ZBZBADD (const int *fold*)

Get an MPI_OP to add binned complex double precision ($Y += X$)

Creates (if it has not already been created) and returns a function handle for an MPI reduction operation that performs the operation $Y += X$ on two arrays of binned complex double precision datatypes of the specified fold. An MPI datatype handle can be created for such a datatype with [binnedMPI_DOUBLE_COMPLEX_BINNED](#).

This method may call `MPI_Op_create()`. If there is an error, this method will call `MPI_Abort()`.

Parameters

<i>fold</i>	the fold of the binned types
-------------	------------------------------

Author

Peter Ahrens

Date

18 Jun 2016

2.4 include/reproBLAS.h File Reference

[reproBLAS.h](#) defines reproducible BLAS Methods.

```
#include <complex.h>
```

Functions

- double [reproBLAS_rdsum](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible sum of double precision vector X.
- double [reproBLAS_rdasum](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible absolute sum of double precision vector X.
- double [reproBLAS_rdnrm2](#) (const int fold, const int N, const double *X, const int incX)
Compute the reproducible Euclidian norm of double precision vector X.
- double [reproBLAS_rddot](#) (const int fold, const int N, const double *X, const int incX, const double *Y, const int incY)
Compute the reproducible dot product of double precision vectors X and Y.
- float [reproBLAS_rsdot](#) (const int fold, const int N, const float *X, const int incX, const float *Y, const int incY)
Compute the reproducible dot product of single precision vectors X and Y.
- float [reproBLAS_rsasum](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible absolute sum of single precision vector X.
- float [reproBLAS_rssum](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible sum of single precision vector X.
- float [reproBLAS_rsnrm2](#) (const int fold, const int N, const float *X, const int incX)
Compute the reproducible Euclidian norm of single precision vector X.
- void [reproBLAS_rzsum_sub](#) (const int fold, const int N, const void *X, int incX, void *sum)
Compute the reproducible sum of complex double precision vector X.
- double [reproBLAS_rdzasum](#) (const int fold, const int N, const void *X, const int incX)
Compute the reproducible absolute sum of complex double precision vector X.
- double [reproBLAS_rdznorm2](#) (const int fold, const int N, const void *X, int incX)
Compute the reproducible Euclidian norm of complex double precision vector X.
- void [reproBLAS_rzdotc_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
Compute the reproducible conjugated dot product of complex double precision vectors X and Y.
- void [reproBLAS_rzdotu_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.
- void [reproBLAS_rcsum_sub](#) (const int fold, const int N, const void *X, const int incX, void *sum)
Compute the reproducible sum of complex single precision vector X.
- float [reproBLAS_rscasum](#) (const int fold, const int N, const void *X, const int incX)
Compute the reproducible absolute sum of complex single precision vector X.

- float [reproBLAS_rscnrm2](#) (const int fold, const int N, const void *X, const int incX)
Compute the reproducible Euclidian norm of complex single precision vector X.
- void [reproBLAS_rcdotc_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
Compute the reproducible conjugated dot product of complex single precision vectors X and Y.
- void [reproBLAS_rcdotu_sub](#) (const int fold, const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.
- void [reproBLAS_rdgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)
Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.
- void [reproBLAS_rdgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)
Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.
- void [reproBLAS_rsgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)
Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.
- void [reproBLAS_rsgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)
Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.
- void [reproBLAS_rzgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)
Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.
- void [reproBLAS_rzgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)
Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.
- void [reproBLAS_rcgemv](#) (const int fold, const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)
Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.
- void [reproBLAS_rcgemm](#) (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)
Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.
- double [reproBLAS_dsum](#) (const int N, const double *X, const int incX)
Compute the reproducible sum of double precision vector X.
- double [reproBLAS_dasum](#) (const int N, const double *X, const int incX)
Compute the reproducible absolute sum of double precision vector X.
- double [reproBLAS_dnrm2](#) (const int N, const double *X, const int incX)
Compute the reproducible Euclidian norm of double precision vector X.
- double [reproBLAS_ddot](#) (const int N, const double *X, const int incX, const double *Y, const int incY)
Compute the reproducible dot product of double precision vectors X and Y.
- float [reproBLAS_sdot](#) (const int N, const float *X, const int incX, const float *Y, const int incY)

- Compute the reproducible dot product of single precision vectors X and Y.*
- float [reproBLAS_sasum](#) (const int N, const float *X, const int incX)
- Compute the reproducible absolute sum of single precision vector X.*
- float [reproBLAS_ssum](#) (const int N, const float *X, const int incX)
- Compute the reproducible sum of single precision vector X.*
- float [reproBLAS_snrm2](#) (const int N, const float *X, const int incX)
- Compute the reproducible Euclidian norm of single precision vector X.*
- void [reproBLAS_zsum_sub](#) (const int N, const void *X, int incX, void *sum)
- Compute the reproducible sum of complex double precision vector X.*
- double [reproBLAS_dzasum](#) (const int N, const void *X, const int incX)
- Compute the reproducible absolute sum of complex double precision vector X.*
- double [reproBLAS_dznrm2](#) (const int N, const void *X, int incX)
- Compute the reproducible Euclidian norm of complex double precision vector X.*
- void [reproBLAS_zdotc_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
- Compute the reproducible conjugated dot product of complex double precision vectors X and Y.*
- void [reproBLAS_zdotu_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
- Compute the reproducible unconjugated dot product of complex double precision vectors X and Y.*
- void [reproBLAS_csum_sub](#) (const int N, const void *X, const int incX, void *sum)
- Compute the reproducible sum of complex single precision vector X.*
- float [reproBLAS_scasum](#) (const int N, const void *X, const int incX)
- Compute the reproducible absolute sum of complex single precision vector X.*
- float [reproBLAS_scnrm2](#) (const int N, const void *X, const int incX)
- Compute the reproducible Euclidian norm of complex single precision vector X.*
- void [reproBLAS_cdotc_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotc)
- Compute the reproducible conjugated dot product of complex single precision vectors X and Y.*
- void [reproBLAS_cdotu_sub](#) (const int N, const void *X, const int incX, const void *Y, const int incY, void *dotu)
- Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.*
- void [reproBLAS_dgemv](#) (const char Order, const char TransA, const int M, const int N, const double alpha, const double *A, const int lda, const double *X, const int incX, const double beta, double *Y, const int incY)
- Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.*
- void [reproBLAS_dgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double *A, const int lda, const double *B, const int ldb, const double beta, double *C, const int ldc)
- Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.*
- void [reproBLAS_sgemv](#) (const char Order, const char TransA, const int M, const int N, const float alpha, const float *A, const int lda, const float *X, const int incX, const float beta, float *Y, const int incY)
- Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.*
- void [reproBLAS_sgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float *A, const int lda, const float *B, const int ldb, const float beta, float *C, const int ldc)
- Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.*
- void [reproBLAS_zgemv](#) (const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)
- Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.*
- void [reproBLAS_zgemm](#) (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

- void `reproBLAS_cgemv` (const char Order, const char TransA, const int M, const int N, const void *alpha, const void *A, const int lda, const void *X, const int incX, const void *beta, void *Y, const int incY)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

- void `reproBLAS_cgemm` (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void *alpha, const void *A, const int lda, const void *B, const int ldb, const void *beta, void *C, const int ldc)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

2.4.1 Detailed Description

`reproBLAS.h` defines reproducible BLAS Methods.

This header is modeled after `cblas.h`, and as such functions are prefixed with character sets describing the data types they operate upon. For example, the function `dfoo` would perform the function `foo` on `double` possibly returning a `double`.

If two character sets are prefixed, the first set of characters describes the output and the second the input type. For example, the function `dzbar` would perform the function `bar` on `double complex` and return a `double`.

Such character sets are listed as follows:

- `d` - double (`double`)
- `z` - complex double (`*void`)
- `s` - float (`float`)
- `c` - complex float (`*void`)

Throughout the library, complex types are specified via `*void` pointers. These routines will sometimes be suffixed by sub, to represent that a function has been made into a subroutine. This allows programmers to use whatever complex types they are already using, as long as the memory pointed to is of the form of two adjacent floating point types, the first and second representing real and imaginary components of the complex number.

The goal of using binned types is to obtain either more accurate or reproducible summation of floating point numbers. In reproducible summation, floating point numbers are split into several slices along predefined boundaries in the exponent range. The space between two boundaries is called a bin. Binned types are composed of several accumulators, each accumulating the slices in a particular bin. The accumulators correspond to the largest consecutive nonzero bins seen so far.

The parameter `fold` describes how many accumulators are used in the binned types supplied to a subroutine (an binned type with `k` accumulators is `k-fold`). The default value for this parameter can be set in `config.h`. If you are unsure of what value to use for `fold`, we recommend 3. Note that the `fold` of binned types must be the same for all binned types that interact with each other. Operations on more than one binned type assume all binned types being operated upon have the same `fold`. Note that the `fold` of an binned type may not be changed once the type has been allocated. A common use case would be to set the value of `fold` as a global macro in your code and supply it to all binned functions that you use.

In `reproBLAS`, two copies of the BLAS are provided. The functions that share the same name as their BLAS counterparts perform reproducible versions of their corresponding operations using the default fold value specified in `config.h`. The functions that are prefixed by the character 'r' allow the user to specify their own fold for the underlying binned types.

2.4.2 Function Documentation

2.4.2.1 void reproBLAS_cdotc_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotc*)

Compute the reproducible conjugated dot product of complex single precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and conjugated *Y*.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_cbcdotc\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.2 void reproBLAS_cdotu_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotu*)

Compute the reproducible unconjugated dot product of complex single precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_cbcdotu\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.3 void reproBLAS_cgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$ or $\text{op}(X) = X^{**H}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with [binnedBLAS_cbcgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar β
<i>C</i>	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.4 void reproBLAS_cgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$ or $y := \alpha * A^H * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with [binnedBLAS_cbcgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.5 void reproBLAS_csum_sub (const int *N*, const void * *X*, const int *incX*, void * *sum*)

Compute the reproducible sum of complex single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using [binnedBLAS_cbcsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.6 double reproBLAS_dasum (const int *N*, const double * *X*, const int *incX*)

Compute the reproducible absolute sum of double precision vector *X*.

Return the sum of absolute values of elements in *X*.

The reproducible absolute sum is computed with binned types of default fold using [binnedBLAS_dbdasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returnsabsolute sum of *X***Author**

Peter Ahrens

Date

15 Jan 2016

2.4.2.7 double reproBLAS_ddot (const int *N*, const double * *X*, const int *incX*, const double * *Y*, const int *incY*)

Compute the reproducible dot product of double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_dbddot\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.8 void reproBLAS_dgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const double *alpha*, const double * *A*, const int *lda*, const double * *B*, const int *ldb*, const double *beta*, double * *C*, const int *ldc*)

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with [binnedBLAS_dbdgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α

Parameters

<i>A</i>	double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar beta
<i>C</i>	double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.9 void reproBLAS_dgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const double *alpha*, const double * *A*, const int *lda*, const double * *X*, const int *incX*, const double *beta*, double * *Y*, const int *incY*)

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with [binnedBLAS_dbdgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.10 double reproBLAS_dnorm2 (const int *N*, const double * *X*, const int *incX*)

Compute the reproducible Euclidian norm of double precision vector *X*.

Return the square root of the sum of the squared elements of *X*.

The reproducible Euclidian norm is computed with scaled binned types of default fold using [binnedBLAS_dbdssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

ReturnsEuclidian norm of *X***Author**

Peter Ahrens

Date

15 Jan 2016

2.4.2.11 double reproBLAS_dsum (const int *N*, const double * *X*, const int *incX*)

Compute the reproducible sum of double precision vector *X*.

Return the sum of *X*.

The reproducible sum is computed with binned types of default fold using [binnedBLAS_dbdsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.12 `double reproBLAS_dzasum (const int N , const void * X , const int $incX$)`

Compute the reproducible absolute sum of complex double precision vector X .

Return the sum of magnitudes of elements of X .

The reproducible absolute sum is computed with binned types of default fold using [binnedBLAS_dbzasum\(\)](#)

Parameters

N	vector length
X	complex double precision vector
$incX$	X vector stride (use every $incX$ 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.13 `double reproBLAS_dznrm2 (const int N , const void * X , const int $incX$)`

Compute the reproducible Euclidian norm of complex double precision vector X .

Return the square root of the sum of the squared elements of X .

The reproducible Euclidian norm is computed with scaled binned types of default fold using [binnedBLAS_dbzssq\(\)](#)

Parameters

N	vector length
X	complex double precision vector
$incX$	X vector stride (use every $incX$ 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.14 `void reproBLAS_rcdotc_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotc)`

Compute the reproducible conjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types using [binnedBLAS_cbcdotc\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.15 `void reproBLAS_rcdotu_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotu)`

Compute the reproducible unconjugated dot product of complex single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using [binnedBLAS_cbcdotu\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>Y</i>	complex single precision vector
<i>incY</i>	Y vector stride (use every incY'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.16 `void reproBLAS_rcgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const void * alpha, const void * A, const int lda, const void * B, const int ldb, const void * beta, void * C, const int ldc)`

Add to complex single precision matrix C the reproducible matrix-matrix product of complex single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with [binnedBLAS_cbcgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar alpha

Parameters

<i>A</i>	complex single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar beta
<i>C</i>	complex single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.17 void reproBLAS_rcgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex single precision vector Y the reproducible matrix-vector product of complex single precision matrix A and complex single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**T} * x + \beta * y$ or $y := \alpha * A^{**H} * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with [binnedBLAS_cbcgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.18 `void reproBLAS_rcsum_sub (const int fold, const int N, const void * X, const int incX, void * sum)`

Compute the reproducible sum of complex single precision vector *X*.

Return the sum of *X*.

The reproducible sum is computed with binned types using [binnedBLAS_cbcsum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.19 `double reproBLAS_rdasum (const int fold, const int N, const double * X, const int incX)`

Compute the reproducible absolute sum of double precision vector *X*.

Return the sum of absolute values of elements in *X*.

The reproducible absolute sum is computed with binned types using [binnedBLAS_dbdasum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.20 `double reproBLAS_rddot (const int fold, const int N, const double * X, const int incX, const double * Y, const int incY)`

Compute the reproducible dot product of double precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using [binnedBLAS_dbddot\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.21 `void reproBLAS_rdgemm (const int fold, const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const double alpha, const double * A, const int lda, const double * B, const int ldb, const double beta, double * C, const int ldc)`

Add to double precision matrix C the reproducible matrix-matrix product of double precision matrices A and B.

Performs one of the matrix-matrix operations

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C,$$

where $\text{op}(X)$ is one of

$$\text{op}(X) = X \text{ or } \text{op}(X) = X^{**T},$$

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with [binnedBLAS_dbdgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C .
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C .
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (ma , lda) in row-major or (lda , na) in column-major. (ma , na) is (M , K) if A is not transposed and (K , M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	double precision matrix of dimension (mb , ldb) in row-major or (ldb , nb) in column-major. (mb , nb) is (K , N) if B is not transposed and (N , K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar β
<i>C</i>	double precision matrix of dimension (M , ldc) in row-major or (ldc , N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.22 `void reproBLAS_rdgemv (const int fold, const char Order, const char TransA, const int M, const int N, const double alpha, const double * A, const int lda, const double * X, const int incX, const double beta, double * Y, const int incY)`

Add to double precision vector Y the reproducible matrix-vector product of double precision matrix A and double precision vector X .

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with [binnedBLAS_dbdgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar α
<i>A</i>	double precision matrix of dimension (M , lda) in row-major or (lda , N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every $incX$ 'th element)
<i>beta</i>	scalar β
<i>Y</i>	double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every $incY$ 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.23 double reproBLAS_rdnrm2 (const int *fold*, const int *N*, const double * *X*, const int *incX*)

Compute the reproducible Euclidian norm of double precision vector X .

Return the square root of the sum of the squared elements of X .

The reproducible Euclidian norm is computed with scaled binned types using [binnedBLAS_dbdssq\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every $incX$ 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.24 `double reproBLAS_rdsun (const int fold, const int N, const double * X, const int incX)`

Compute the reproducible sum of double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using [binnedBLAS_dbdsum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.25 `double reproBLAS_rdzasum (const int fold, const int N, const void * X, const int incX)`

Compute the reproducible absolute sum of complex double precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types using [binnedBLAS_dbzasum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.26 `double reproBLAS_rdznm2 (const int fold, const int N, const void * X, const int incX)`

Compute the reproducible Euclidian norm of complex double precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using [binnedBLAS_dbzssq\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.27 `float reproBLAS_rsasum (const int fold, const int N, const float * X, const int incX)`

Compute the reproducible absolute sum of single precision vector *X*.

Return the sum of absolute values of elements in *X*.

The reproducible absolute sum is computed with binned types using [binnedBLAS_sbsasum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.28 float reproBLAS_rscasum (const int *fold*, const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector X.

Return the sum of magnitudes of elements of X.

The reproducible absolute sum is computed with binned types using [binnedBLAS_sbcasum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.29 `float reproBLAS_rscnrm2 (const int fold, const int N, const void * X, const int incX)`

Compute the reproducible Euclidian norm of complex single precision vector *X*.

Return the square root of the sum of the squared elements of *X*.

The reproducible Euclidian norm is computed with scaled binned types using [binnedBLAS_sbcssq\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.30 float reproBLAS_rsdot (const int *fold*, const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types using [binnedBLAS_sbsdot\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.31 void reproBLAS_rsgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const float *alpha*, const float * *A*, const int *lda*, const float * *B*, const int *ldb*, const float *beta*, float * *C*, const int *ldc*)

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with [binnedBLAS_sbsgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar β
<i>C</i>	single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.32 void reproBLAS_rsgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, const float *beta*, float * *Y*, const int *incY*)

Add to single precision vector Y the reproducible matrix-vector product of single precision matrix A and single precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with [binnedBLAS_sbsgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	single precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	single precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	single precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.33 float reproBLAS_rsnrm2 (const int *fold*, const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types using [binnedBLAS_sbsssq\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every incX'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.34 `float reproBLAS_rssum (const int fold, const int N, const float * X, const int incX)`

Compute the reproducible sum of single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using [binnedBLAS_sbssum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.35 `void reproBLAS_rzdotc_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotc)`

Compute the reproducible conjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types using [binnedBLAS_zbzdotc\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.36 `void reproBLAS_rzdotu_sub (const int fold, const int N, const void * X, const int incX, const void * Y, const int incY, void * dotu)`

Compute the reproducible unconjugated dot product of complex double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with binned types using [binnedBLAS_zbzdotu\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.37 void reproBLAS_rzgemm (const int *fold*, const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex double precision matrix C the reproducible matrix-matrix product of complex double precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where op(X) is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**}T$ or $\text{op}(X) = X^{**}H$,

alpha and beta are scalars, A and B and C are matrices with op(A) an M by K matrix, op(B) a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types with [binnedBLAS_zbzgemm\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix op(A) and of the matrix C.
<i>N</i>	number of columns of matrix op(B) and of the matrix C.
<i>K</i>	number of columns of matrix op(A) and columns of the matrix op(B).
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar beta
<i>C</i>	complex double precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.38 void reproBLAS_rzgemv (const int *fold*, const char *Order*, const char *TransA*, const int *M*, const int *N*, const void * *alpha*, const void * *A*, const int *lda*, const void * *X*, const int *incX*, const void * *beta*, void * *Y*, const int *incY*)

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**T} * x + \beta * y$ or $y := \alpha * A^{**H} * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types with [binnedBLAS_zbzgemv\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.39 void reproBLAS_rzsum_sub (const int *fold*, const int *N*, const void * *X*, const int *incX*, void * *sum*)

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types using [binnedBLAS_zbzsum\(\)](#)

Parameters

<i>fold</i>	the fold of the binned types
<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.40 float reproBLAS_sasum (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible absolute sum of single precision vector *X*.

Return the sum of absolute values of elements in *X*.

The reproducible absolute sum is computed with binned types of default fold using [binnedBLAS_sbsasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returnsabsolute sum of *X***Author**

Peter Ahrens

Date

15 Jan 2016

2.4.2.41 float reproBLAS_scasum (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible absolute sum of complex single precision vector *X*.

Return the sum of magnitudes of elements of *X*.

The reproducible absolute sum is computed with binned types of default fold using [binnedBLAS_sbcasum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)

Returns

absolute sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.42 float reproBLAS_scnrm2 (const int *N*, const void * *X*, const int *incX*)

Compute the reproducible Euclidian norm of complex single precision vector X.

Return the square root of the sum of the squared elements of X.

The reproducible Euclidian norm is computed with scaled binned types of default fold using [binnedBLAS_sbcssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.43 float reproBLAS_sdot (const int *N*, const float * *X*, const int *incX*, const float * *Y*, const int *incY*)

Compute the reproducible dot product of single precision vectors X and Y.

Return the sum of the pairwise products of X and Y.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_sbsdot\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	single precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)

Returns

the dot product of X and Y

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.44 `void reproBLAS_sgemm (const char Order, const char TransA, const char TransB, const int M, const int N, const int K, const float alpha, const float * A, const int lda, const float * B, const int ldb, const float beta, float * C, const int ldc)`

Add to single precision matrix C the reproducible matrix-matrix product of single precision matrices A and B.

Performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$,

where $\text{op}(X)$ is one of

$\text{op}(X) = X$ or $\text{op}(X) = X^{**T}$,

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with [binnedBLAS_sbsgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C.
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C.
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	single precision matrix of dimension (ma, lda) in row-major or (lda, na) in column-major. (ma, na) is (M, K) if A is not transposed and (K, M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	single precision matrix of dimension (mb, ldb) in row-major or (ldb, nb) in column-major. (mb, nb) is (K, N) if B is not transposed and (N, K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar β
<i>C</i>	single precision matrix of dimension (M, ldc) in row-major or (ldc, N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.45 void reproBLAS_sgemv (const char *Order*, const char *TransA*, const int *M*, const int *N*, const float *alpha*, const float * *A*, const int *lda*, const float * *X*, const int *incX*, const float *beta*, float * *Y*, const int *incY*)

Add to single precision vector *Y* the reproducible matrix-vector product of single precision matrix *A* and single precision vector *X*.

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^T * x + \beta * y$,

where α and β are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with [binnedBLAS_sbssgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose <i>A</i> before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' or 'c' or 'C' to transpose)
<i>M</i>	number of rows of matrix <i>A</i>
<i>N</i>	number of columns of matrix <i>A</i>
<i>alpha</i>	scalar α
<i>A</i>	single precision matrix of dimension (<i>M</i> , <i>lda</i>) in row-major or (<i>lda</i> , <i>N</i>) in column-major
<i>lda</i>	the first dimension of <i>A</i> as declared in the calling program
<i>X</i>	single precision vector of at least size <i>N</i> if not transposed or size <i>M</i> otherwise
<i>incX</i>	<i>X</i> vector stride (use every <i>incX</i> 'th element)
<i>beta</i>	scalar β
<i>Y</i>	single precision vector <i>Y</i> of at least size <i>M</i> if not transposed or size <i>N</i> otherwise
<i>incY</i>	<i>Y</i> vector stride (use every <i>incY</i> 'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.46 float reproBLAS_snrm2 (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible Euclidian norm of single precision vector *X*.

Return the square root of the sum of the squared elements of *X*.

The reproducible Euclidian norm is computed with scaled binned types of default fold using [binnedBLAS_sbsssq\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

Euclidian norm of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.47 float reproBLAS_ssum (const int *N*, const float * *X*, const int *incX*)

Compute the reproducible sum of single precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using [binnedBLAS_sbssum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	single precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)

Returns

sum of X

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.48 void reproBLAS_zdotc_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotc*)

Compute the reproducible conjugated dot product of complex double precision vectors X and Y.

Return the sum of the pairwise products of X and conjugated Y.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_zbzdadc\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotc</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.49 void reproBLAS_zdotu_sub (const int *N*, const void * *X*, const int *incX*, const void * *Y*, const int *incY*, void * *dotu*)

Compute the reproducible unconjugated dot product of complex double precision vectors *X* and *Y*.

Return the sum of the pairwise products of *X* and *Y*.

The reproducible dot product is computed with binned types of default fold using [binnedBLAS_zbzdotu\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every <i>incX</i> 'th element)
<i>Y</i>	complex double precision vector
<i>incY</i>	Y vector stride (use every <i>incY</i> 'th element)
<i>dotu</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

2.4.2.50 void reproBLAS_zgemm (const char *Order*, const char *TransA*, const char *TransB*, const int *M*, const int *N*, const int *K*, const void * *alpha*, const void * *A*, const int *lda*, const void * *B*, const int *ldb*, const void * *beta*, void * *C*, const int *ldc*)

Add to complex double precision matrix *C* the reproducible matrix-matrix product of complex double precision matrices *A* and *B*.

Performs one of the matrix-matrix operations

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C,$$

where $\text{op}(X)$ is one of

$$\text{op}(X) = X \text{ or } \text{op}(X) = X^{**T} \text{ or } \text{op}(X) = X^{**H},$$

α and β are scalars, A and B and C are matrices with $\text{op}(A)$ an M by K matrix, $\text{op}(B)$ a K by N matrix, and C is an M by N matrix.

The matrix-matrix product is computed using binned types of default fold with [binnedBLAS_zbzgemm\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>TransB</i>	a character specifying whether or not to transpose B before taking the matrix-matrix product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix $\text{op}(A)$ and of the matrix C .
<i>N</i>	number of columns of matrix $\text{op}(B)$ and of the matrix C .
<i>K</i>	number of columns of matrix $\text{op}(A)$ and columns of the matrix $\text{op}(B)$.
<i>alpha</i>	scalar α
<i>A</i>	complex double precision matrix of dimension (ma , lda) in row-major or (lda , na) in column-major. (ma , na) is (M , K) if A is not transposed and (K , M) otherwise.
<i>lda</i>	the first dimension of A as declared in the calling program. lda must be at least na in row major or ma in column major.
<i>B</i>	complex double precision matrix of dimension (mb , ldb) in row-major or (ldb , nb) in column-major. (mb , nb) is (K , N) if B is not transposed and (N , K) otherwise.
<i>ldb</i>	the first dimension of B as declared in the calling program. ldb must be at least nb in row major or mb in column major.
<i>beta</i>	scalar β
<i>C</i>	complex double precision matrix of dimension (M , ldc) in row-major or (ldc , N) in column-major.
<i>ldc</i>	the first dimension of C as declared in the calling program. ldc must be at least N in row major or M in column major.

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.51 `void reproBLAS_zgemv (const char Order, const char TransA, const int M, const int N, const void * alpha, const void * A, const int lda, const void * X, const int incX, const void * beta, void * Y, const int incY)`

Add to complex double precision vector Y the reproducible matrix-vector product of complex double precision matrix A and complex double precision vector X .

Performs one of the matrix-vector operations

$y := \alpha * A * x + \beta * y$ or $y := \alpha * A^{**T} * x + \beta * y$ or $y := \alpha * A^{**H} * x + \beta * y$,

where alpha and beta are scalars, x and y are vectors, and A is an M by N matrix.

The matrix-vector product is computed using binned types of default fold with [binnedBLAS_zbzgemv\(\)](#)

Parameters

<i>Order</i>	a character specifying the matrix ordering ('r' or 'R' for row-major, 'c' or 'C' for column major)
<i>TransA</i>	a character specifying whether or not to transpose A before taking the matrix-vector product ('n' or 'N' not to transpose, 't' or 'T' to transpose, 'c' or 'C' to conjugate transpose)
<i>M</i>	number of rows of matrix A
<i>N</i>	number of columns of matrix A
<i>alpha</i>	scalar alpha
<i>A</i>	complex double precision matrix of dimension (M, lda) in row-major or (lda, N) in column-major
<i>lda</i>	the first dimension of A as declared in the calling program
<i>X</i>	complex double precision vector of at least size N if not transposed or size M otherwise
<i>incX</i>	X vector stride (use every incX'th element)
<i>beta</i>	scalar beta
<i>Y</i>	complex double precision vector Y of at least size M if not transposed or size N otherwise
<i>incY</i>	Y vector stride (use every incY'th element)

Author

Peter Ahrens

Date

18 Jan 2016

2.4.2.52 `void reproBLAS_zsum_sub (const int N, const void * X, const int incX, void * sum)`

Compute the reproducible sum of complex double precision vector X.

Return the sum of X.

The reproducible sum is computed with binned types of default fold using [binnedBLAS_zbzsum\(\)](#)

Parameters

<i>N</i>	vector length
<i>X</i>	complex double precision vector
<i>incX</i>	X vector stride (use every incX'th element)
<i>sum</i>	scalar return

Author

Peter Ahrens

Date

15 Jan 2016

Index

binmed.h

binmed_DBCAPACITY, 11
binmed_DBENDURANCE, 11
binmed_DBMAXFOLD, 11
binmed_DBMAXINDEX, 12
binmed_DMCOMPRESSION, 12
binmed_DMEXPANSION, 12
binmed_SBCAPACITY, 12
binmed_SBENDURANCE, 13
binmed_SBMAXFOLD, 13
binmed_SBMAXINDEX, 13
binmed_SMCOMPRESSION, 14
binmed_SMEXPANSION, 14
binmed_cballoc, 16
binmed_cbcadd, 17
binmed_cbcadd, 17
binmed_cbcadd, 18
binmed_cbcadd, 18
binmed_cbcadd, 19
binmed_cbcadd, 19
binmed_cbcadd, 20
binmed_cbcadd, 20
binmed_cbcadd, 20
binmed_cbcadd, 21
binmed_cbcadd, 21
binmed_cbcadd, 22
binmed_cbcadd, 22
binmed_cbcadd, 22
binmed_cbcadd, 23
binmed_cbcadd, 23
binmed_cbcadd, 24
binmed_cbcadd, 24
binmed_cbcadd, 25
binmed_cbcadd, 25
binmed_cbcadd, 26
binmed_cbcadd, 26
binmed_cbcadd, 27
binmed_cbcadd, 27
binmed_cbcadd, 28
binmed_cbcadd, 28
binmed_cbcadd, 29
binmed_cbcadd, 29
binmed_cbcadd, 30
binmed_cbcadd, 30
binmed_cbcadd, 32
binmed_cbcadd, 32
binmed_cbcadd, 33
binmed_cbcadd, 33
binmed_cbcadd, 34

binmed_dbdbaddsq, 34
binmed_dbdbaddv, 35
binmed_dbdbaddv, 35
binmed_dbdbaddv, 36
binmed_dbdbaddv, 36
binmed_dbdbaddv, 37
binmed_dbdbaddv, 37
binmed_dbdbaddv, 37
binmed_dbdbaddv, 38
binmed_dbdbaddv, 38
binmed_dbdbaddv, 39
binmed_dbdbaddv, 39
binmed_dbdbaddv, 40
binmed_dbdbaddv, 40
binmed_dbdbaddv, 41
binmed_dbdbaddv, 41
binmed_dbdbaddv, 42
binmed_dbdbaddv, 42
binmed_dbdbaddv, 43
binmed_dbdbaddv, 44
binmed_dbdbaddv, 44
binmed_dbdbaddv, 45
binmed_dbdbaddv, 45
binmed_dbdbaddv, 46
binmed_dbdbaddv, 46
binmed_dbdbaddv, 47
binmed_dbdbaddv, 47
binmed_dbdbaddv, 48
binmed_dbdbaddv, 48
binmed_dbdbaddv, 49
binmed_dbdbaddv, 49
binmed_dbdbaddv, 50
binmed_dbdbaddv, 50
binmed_dbdbaddv, 51
binmed_dbdbaddv, 51
binmed_dbdbaddv, 52
binmed_dbdbaddv, 52
binmed_dbdbaddv, 53
binmed_dbdbaddv, 53
binmed_dbdbaddv, 54
binmed_dbdbaddv, 55
binmed_dbdbaddv, 55
binmed_dbdbaddv, 56
binmed_dbdbaddv, 56
binmed_dbdbaddv, 57
binmed_dbdbaddv, 57

binned_smbins, 58
 binned_smdenorm, 58
 binned_smindex, 59
 binned_smindex0, 59
 binned_smnegate, 60
 binned_smprint, 60
 binned_smrenorm, 61
 binned_smsadd, 61
 binned_smsconv, 62
 binned_smsdeposit, 62
 binned_smsetzero, 63
 binned_smsmadd, 63
 binned_smsmaddsq, 64
 binned_smsmset, 64
 binned_smsrescale, 65
 binned_smsupdate, 65
 binned_ssbconv, 67
 binned_sscale, 67
 binned_ssmconv, 68
 binned_ufp, 68
 binned_ufpf, 69
 binned_zballoc, 69
 binned_zbdbset, 70
 binned_zbduupdate, 70
 binned_zbnegate, 71
 binned_zbnum, 71
 binned_zbprint, 71
 binned_zbrenorm, 72
 binned_zbsetzero, 72
 binned_zbsize, 73
 binned_zbzadd, 73
 binned_zbzbadd, 74
 binned_zbzbaddv, 74
 binned_zbzbset, 74
 binned_zbzconv, 75
 binned_zbzdeposit, 75
 binned_zbzupdate, 77
 binned_zmdenorm, 77
 binned_zmdmset, 78
 binned_zmdrescale, 78
 binned_zmdupdate, 79
 binned_zmnegate, 79
 binned_zmprint, 80
 binned_zmrenorm, 80
 binned_zmsetzero, 81
 binned_zmzadd, 81
 binned_zmzconv, 82
 binned_zmzdeposit, 82
 binned_zmzmadd, 83
 binned_zmzmset, 83
 binned_zmzupdate, 84
 binned_zzbconv_sub, 84
 binned_zzmconv_sub, 85
 DBWIDTH, 14
 double_binned, 15
 double_complex_binned, 15
 float_binned, 16
 float_complex_binned, 16

SBWIDTH, 15
 binned_DBCAPACITY
 binned.h, 11
 binned_DBENDURANCE
 binned.h, 11
 binned_DBMAXFOLD
 binned.h, 11
 binned_DBMAXINDEX
 binned.h, 12
 binned_DMCOMPRESSION
 binned.h, 12
 binned_DMEXPANSION
 binned.h, 12
 binned_SBCAPACITY
 binned.h, 12
 binned_SBENDURANCE
 binned.h, 13
 binned_SBMAXFOLD
 binned.h, 13
 binned_SBMAXINDEX
 binned.h, 13
 binned_SMCOMPRESSION
 binned.h, 14
 binned_SMEXPANSION
 binned.h, 14
 binned_cballoc
 binned.h, 16
 binned_cbcadd
 binned.h, 17
 binned_cbcadd
 binned.h, 17
 binned_cbcaddv
 binned.h, 18
 binned_cbcset
 binned.h, 18
 binned_cbconv
 binned.h, 19
 binned_cbcdeposit
 binned.h, 19
 binned_cbcupdate
 binned.h, 20
 binned_cbnegate
 binned.h, 20
 binned_cbnum
 binned.h, 20
 binned_cbprint
 binned.h, 21
 binned_cbrenorm
 binned.h, 21
 binned_cbsbset
 binned.h, 22
 binned_cbsetzero
 binned.h, 22
 binned_cbsize
 binned.h, 22
 binned_cbsupdate
 binned.h, 23
 binned_ccbconv_sub

binned.h, [23](#)
binned_ccmconv_sub
 binned.h, [24](#)
binned_cmcadd
 binned.h, [24](#)
binned_cmconv
 binned.h, [25](#)
binned_cmcdposit
 binned.h, [25](#)
binned_cmcmadd
 binned.h, [26](#)
binned_cmcmset
 binned.h, [26](#)
binned_cmcupdate
 binned.h, [27](#)
binned_cmddenorm
 binned.h, [27](#)
binned_cmnegate
 binned.h, [28](#)
binned_cmprint
 binned.h, [28](#)
binned_cmrenorm
 binned.h, [29](#)
binned_cmsetzero
 binned.h, [29](#)
binned_cmsmset
 binned.h, [30](#)
binned_cmsrescale
 binned.h, [30](#)
binned_cmsupdate
 binned.h, [32](#)
binned_dballoc
 binned.h, [32](#)
binned_dbbound
 binned.h, [33](#)
binned_dbdadd
 binned.h, [33](#)
binned_dbdbadd
 binned.h, [34](#)
binned_dbdbaddsq
 binned.h, [34](#)
binned_dbdbaddv
 binned.h, [35](#)
binned_dbdbset
 binned.h, [35](#)
binned_dbdconv
 binned.h, [36](#)
binned_dbddeposit
 binned.h, [36](#)
binned_dbdupdate
 binned.h, [37](#)
binned_dbnegate
 binned.h, [37](#)
binned_dbnum
 binned.h, [37](#)
binned_dbprint
 binned.h, [38](#)
binned_dbrenorm
 binned.h, [38](#)
binned_dbsetzero
 binned.h, [39](#)
binned_dbsize
 binned.h, [39](#)
binned_ddbconv
 binned.h, [39](#)
binned_ddmconv
 binned.h, [40](#)
binned_dindex
 binned.h, [40](#)
binned_dmbins
 binned.h, [41](#)
binned_dmdadd
 binned.h, [41](#)
binned_dmdconv
 binned.h, [42](#)
binned_dmddeposit
 binned.h, [42](#)
binned_dmdenorm
 binned.h, [43](#)
binned_dmdmadd
 binned.h, [44](#)
binned_dmdmaddsq
 binned.h, [44](#)
binned_dmdmset
 binned.h, [45](#)
binned_dmdrescale
 binned.h, [45](#)
binned_dmdupdate
 binned.h, [46](#)
binned_dmindex
 binned.h, [46](#)
binned_dmindex0
 binned.h, [47](#)
binned_dmnegate
 binned.h, [47](#)
binned_dmprint
 binned.h, [48](#)
binned_dmrenorm
 binned.h, [48](#)
binned_dmsetzero
 binned.h, [49](#)
binned_dscale
 binned.h, [49](#)
binned_sballot
 binned.h, [50](#)
binned_sbbound
 binned.h, [50](#)
binned_sbnegate
 binned.h, [51](#)
binned_sbnun
 binned.h, [51](#)
binned_sbprint
 binned.h, [52](#)
binned_sbreunorm
 binned.h, [52](#)
binned_sbsadd

binned.h, [53](#)
binned_sbsbadd
 binned.h, [53](#)
binned_sbsbaddsq
 binned.h, [53](#)
binned_sbsbaddv
 binned.h, [54](#)
binned_sbsbset
 binned.h, [55](#)
binned_sbsbze
 binned.h, [55](#)
binned_sbsconv
 binned.h, [55](#)
binned_sbsdeposit
 binned.h, [56](#)
binned_sbsetzero
 binned.h, [56](#)
binned_sbsupdate
 binned.h, [57](#)
binned_sindex
 binned.h, [57](#)
binned_smbins
 binned.h, [58](#)
binned_smdenorm
 binned.h, [58](#)
binned_smindex
 binned.h, [59](#)
binned_smindex0
 binned.h, [59](#)
binned_smnegate
 binned.h, [60](#)
binned_smprint
 binned.h, [60](#)
binned_smrenorm
 binned.h, [61](#)
binned_smsadd
 binned.h, [61](#)
binned_smsconv
 binned.h, [62](#)
binned_smsdeposit
 binned.h, [62](#)
binned_smsetzero
 binned.h, [63](#)
binned_smsmadd
 binned.h, [63](#)
binned_smsmaddsq
 binned.h, [64](#)
binned_smsmset
 binned.h, [64](#)
binned_smsrescale
 binned.h, [65](#)
binned_smsupdate
 binned.h, [65](#)
binned_ssbconv
 binned.h, [67](#)
binned_sscale
 binned.h, [67](#)
binned_ssmconv
 binned.h, [68](#)
binned_ufp
 binned.h, [68](#)
binned_ufpf
 binned.h, [69](#)
binned_zballoc
 binned.h, [69](#)
binned_zbdbset
 binned.h, [70](#)
binned_zbupdate
 binned.h, [70](#)
binned_zbnegate
 binned.h, [71](#)
binned_zbnum
 binned.h, [71](#)
binned_zbprint
 binned.h, [71](#)
binned_zbrenorm
 binned.h, [72](#)
binned_zbsetzero
 binned.h, [72](#)
binned_zbsize
 binned.h, [73](#)
binned_zbzadd
 binned.h, [73](#)
binned_zbzbadd
 binned.h, [74](#)
binned_zbzbaddv
 binned.h, [74](#)
binned_zbzbset
 binned.h, [74](#)
binned_zbzconv
 binned.h, [75](#)
binned_zbzdeposit
 binned.h, [75](#)
binned_zbzupdate
 binned.h, [77](#)
binned_zmdenorm
 binned.h, [77](#)
binned_zmdmset
 binned.h, [78](#)
binned_zmdrescale
 binned.h, [78](#)
binned_zmdupdate
 binned.h, [79](#)
binned_zmnegate
 binned.h, [79](#)
binned_zmprint
 binned.h, [80](#)
binned_zmrenorm
 binned.h, [80](#)
binned_zmsetzero
 binned.h, [81](#)
binned_zmzadd
 binned.h, [81](#)
binned_zmzconv
 binned.h, [82](#)
binned_zmzdeposit

Generated by Doxygen

reproBLAS_dnrm2, 137
reproBLAS_dsum, 137
reproBLAS_dzasum, 138
reproBLAS_dznrm2, 138
reproBLAS_rcdotc_sub, 139
reproBLAS_rcdotu_sub, 139
reproBLAS_rcgemm, 140
reproBLAS_rcgemv, 141
reproBLAS_rcsum_sub, 142
reproBLAS_rdasum, 142
reproBLAS_rddot, 143
reproBLAS_rdgemm, 143
reproBLAS_rdgemv, 144
reproBLAS_rdnrm2, 145
reproBLAS_rdsun, 146
reproBLAS_rdzasum, 146
reproBLAS_rdnrm2, 147
reproBLAS_rsasum, 147
reproBLAS_rscasum, 149
reproBLAS_rscnrm2, 149
reproBLAS_rsdot, 151
reproBLAS_rsgemm, 151
reproBLAS_rsgemv, 152
reproBLAS_rsnrm2, 153
reproBLAS_rssum, 154
reproBLAS_rzdotc_sub, 154
reproBLAS_rzdotu_sub, 155
reproBLAS_rzgemm, 155
reproBLAS_rzgemv, 156
reproBLAS_rzsum_sub, 157
reproBLAS_sasum, 158
reproBLAS_scasum, 158
reproBLAS_scnrm2, 159
reproBLAS_sdot, 159
reproBLAS_sgemm, 160
reproBLAS_sgemv, 161
reproBLAS_snrm2, 161
reproBLAS_ssum, 162
reproBLAS_zdotc_sub, 162
reproBLAS_zdotu_sub, 163
reproBLAS_zgemm, 163
reproBLAS_zgemv, 164
reproBLAS_zsum_sub, 166
reproBLAS_cdotc_sub
 reproBLAS.h, 131
reproBLAS_cdotu_sub
 reproBLAS.h, 131
reproBLAS_cgemm
 reproBLAS.h, 132
reproBLAS_cgemv
 reproBLAS.h, 133
reproBLAS_csum_sub
 reproBLAS.h, 133
reproBLAS_dasum
 reproBLAS.h, 134
reproBLAS_ddot
 reproBLAS.h, 134
reproBLAS_dgemm
 reproBLAS.h, 135
reproBLAS_dgemv
 reproBLAS.h, 136
reproBLAS_dnrm2
 reproBLAS.h, 137
reproBLAS_dsum
 reproBLAS.h, 137
reproBLAS_dzasum
 reproBLAS.h, 138
reproBLAS_dznrm2
 reproBLAS.h, 138
reproBLAS_rcdotc_sub
 reproBLAS.h, 139
reproBLAS_rcdotu_sub
 reproBLAS.h, 139
reproBLAS_rcgemm
 reproBLAS.h, 140
reproBLAS_rcgemv
 reproBLAS.h, 141
reproBLAS_rcsum_sub
 reproBLAS.h, 142
reproBLAS_rdasum
 reproBLAS.h, 142
reproBLAS_rddot
 reproBLAS.h, 143
reproBLAS_rdgemm
 reproBLAS.h, 143
reproBLAS_rdgemv
 reproBLAS.h, 144
reproBLAS_rdnrm2
 reproBLAS.h, 145
reproBLAS_rdsun
 reproBLAS.h, 146
reproBLAS_rdzasum
 reproBLAS.h, 146
reproBLAS_rdnrm2
 reproBLAS.h, 147
reproBLAS_rsasum
 reproBLAS.h, 147
reproBLAS_rscasum
 reproBLAS.h, 149
reproBLAS_rscnrm2
 reproBLAS.h, 149
reproBLAS_rsdot
 reproBLAS.h, 151
reproBLAS_rsgemm
 reproBLAS.h, 151
reproBLAS_rsgemv
 reproBLAS.h, 152
reproBLAS_rsnrm2
 reproBLAS.h, 153
reproBLAS_rssum
 reproBLAS.h, 154
reproBLAS_rzdotc_sub
 reproBLAS.h, 154
reproBLAS_rzdotu_sub
 reproBLAS.h, 155
reproBLAS_rzgemm

reproBLAS.h, [155](#)
reproBLAS_rzgemv
 reproBLAS.h, [156](#)
reproBLAS_rzsum_sub
 reproBLAS.h, [157](#)
reproBLAS_sasum
 reproBLAS.h, [158](#)
reproBLAS_scasum
 reproBLAS.h, [158](#)
reproBLAS_scnrm2
 reproBLAS.h, [159](#)
reproBLAS_sdot
 reproBLAS.h, [159](#)
reproBLAS_sgemm
 reproBLAS.h, [160](#)
reproBLAS_sgemv
 reproBLAS.h, [161](#)
reproBLAS_snrm2
 reproBLAS.h, [161](#)
reproBLAS_ssum
 reproBLAS.h, [162](#)
reproBLAS_zdotc_sub
 reproBLAS.h, [162](#)
reproBLAS_zdotu_sub
 reproBLAS.h, [163](#)
reproBLAS_zgemm
 reproBLAS.h, [163](#)
reproBLAS_zgemv
 reproBLAS.h, [164](#)
reproBLAS_zsum_sub
 reproBLAS.h, [166](#)

SBWIDTH
 binned.h, [15](#)