

# Fast Auxiliary Space Preconditioning

2.7.0 Aug/10/2021

Generated by Doxygen 1.9.1



<b>1 Introduction</b>	<b>1</b>
<b>2 How to obtain FASP</b>	<b>3</b>
<b>3 Building and Installation</b>	<b>5</b>
<b>4 Developers</b>	<b>7</b>
<b>5 Doxygen</b>	<b>9</b>
<b>6 Data Structure Index</b>	<b>11</b>
6.1 Data Structures . . . . .	11
<b>7 File Index</b>	<b>13</b>
7.1 File List . . . . .	13
<b>8 Data Structure Documentation</b>	<b>17</b>
8.1 AMG_data Struct Reference . . . . .	17
8.1.1 Detailed Description . . . . .	18
8.2 AMG_data_bsr Struct Reference . . . . .	18
8.2.1 Detailed Description . . . . .	20
8.3 AMG_param Struct Reference . . . . .	20
8.3.1 Detailed Description . . . . .	22
8.4 block_dvector Struct Reference . . . . .	23
8.4.1 Detailed Description . . . . .	23
8.5 block_ivector Struct Reference . . . . .	23
8.5.1 Detailed Description . . . . .	23
8.6 dBLCmat Struct Reference . . . . .	24
8.6.1 Detailed Description . . . . .	24
8.7 dBSRmat Struct Reference . . . . .	24
8.7.1 Detailed Description . . . . .	25
8.7.2 Field Documentation . . . . .	25
8.7.2.1 JA . . . . .	25
8.7.2.2 val . . . . .	25
8.8 dCOOmat Struct Reference . . . . .	26
8.8.1 Detailed Description . . . . .	26
8.9 dCSRLmat Struct Reference . . . . .	26
8.9.1 Detailed Description . . . . .	27
8.10 dCSRmat Struct Reference . . . . .	27
8.10.1 Detailed Description . . . . .	28
8.11 ddenmat Struct Reference . . . . .	28
8.11.1 Detailed Description . . . . .	28

8.12 dSTRmat Struct Reference	29
8.12.1 Detailed Description	29
8.13 dvector Struct Reference	30
8.13.1 Detailed Description	30
8.14 grid2d Struct Reference	30
8.14.1 Detailed Description	31
8.14.2 Field Documentation	31
8.14.2.1 e	31
8.14.2.2 edges	31
8.14.2.3 ediri	31
8.14.2.4 efather	32
8.14.2.5 p	32
8.14.2.6 pdiri	32
8.14.2.7 pfather	32
8.14.2.8 s	32
8.14.2.9 t	33
8.14.2.10 tfather	33
8.14.2.11 triangles	33
8.14.2.12 vertices	33
8.15 iBLCmat Struct Reference	33
8.15.1 Detailed Description	34
8.16 iCOOmat Struct Reference	34
8.16.1 Detailed Description	35
8.17 iCSRmat Struct Reference	35
8.17.1 Detailed Description	35
8.18 idenmat Struct Reference	36
8.18.1 Detailed Description	36
8.19 ILU_data Struct Reference	36
8.19.1 Detailed Description	38
8.20 ILU_param Struct Reference	38
8.20.1 Detailed Description	38
8.21 input_param Struct Reference	39
8.21.1 Detailed Description	40
8.21.2 Field Documentation	40
8.21.2.1 AMG_aggregation_type	40
8.21.2.2 AMG_aggressive_level	40
8.21.2.3 AMG_aggressive_path	41
8.21.2.4 AMG_amli_degree	41
8.21.2.5 AMG_coarse_dof	41

---

8.21.2.6 AMG_coarse_scaling . . . . .	41
8.21.2.7 AMG_coarse_solver . . . . .	41
8.21.2.8 AMG_coarsening_type . . . . .	42
8.21.2.9 AMG_cycle_type . . . . .	42
8.21.2.10 AMG_ILU_levels . . . . .	42
8.21.2.11 AMG_interpolation_type . . . . .	42
8.21.2.12 AMG_levels . . . . .	42
8.21.2.13 AMG_max_aggregation . . . . .	43
8.21.2.14 AMG_max_row_sum . . . . .	43
8.21.2.15 AMG_maxit . . . . .	43
8.21.2.16 AMG_nl_amli_krylov_type . . . . .	43
8.21.2.17 AMG_pair_number . . . . .	43
8.21.2.18 AMG_polynomial_degree . . . . .	44
8.21.2.19 AMG_postsmooth_iter . . . . .	44
8.21.2.20 AMG_presmooth_iter . . . . .	44
8.21.2.21 AMG_quality_bound . . . . .	44
8.21.2.22 AMG_relaxation . . . . .	44
8.21.2.23 AMG_smooth_filter . . . . .	45
8.21.2.24 AMG_smooth_order . . . . .	45
8.21.2.25 AMG_smooth_restriction . . . . .	45
8.21.2.26 AMG_smoother . . . . .	45
8.21.2.27 AMG_strong_coupled . . . . .	45
8.21.2.28 AMG_strong_threshold . . . . .	46
8.21.2.29 AMG_SWZ_levels . . . . .	46
8.21.2.30 AMG_tentative_smooth . . . . .	46
8.21.2.31 AMG_tol . . . . .	46
8.21.2.32 AMG_truncation_threshold . . . . .	46
8.21.2.33 AMG_type . . . . .	47
8.21.2.34 decoup_type . . . . .	47
8.21.2.35 ILU_droptol . . . . .	47
8.21.2.36 ILU_lfil . . . . .	47
8.21.2.37 ILU_permtol . . . . .	47
8.21.2.38 ILU_relax . . . . .	48
8.21.2.39 ILU_type . . . . .	48
8.21.2.40 inifile . . . . .	48
8.21.2.41 itsolver_maxit . . . . .	48
8.21.2.42 itsolver_tol . . . . .	48
8.21.2.43 output_type . . . . .	49
8.21.2.44 precondition_type . . . . .	49

8.21.2.45 print_level	49
8.21.2.46 problem_num	49
8.21.2.47 restart	49
8.21.2.48 solver_type	50
8.21.2.49 stop_type	50
8.21.2.50 SWZ_blksolver	50
8.21.2.51 SWZ_maxlvl	50
8.21.2.52 SWZ_mmsize	50
8.21.2.53 SWZ_type	51
8.21.2.54 workdir	51
8.22 ITS_param Struct Reference	51
8.22.1 Detailed Description	51
8.22.2 Field Documentation	52
8.22.2.1 decoup_type	52
8.22.2.2 itsolver_type	52
8.22.2.3 maxit	52
8.22.2.4 precondition_type	52
8.22.2.5 print_level	53
8.22.2.6 restart	53
8.22.2.7 stop_type	53
8.22.2.8 tol	53
8.23 ivector Struct Reference	53
8.23.1 Detailed Description	54
8.24 Mumps_data Struct Reference	54
8.24.1 Detailed Description	54
8.25 mxv_matfree Struct Reference	55
8.25.1 Detailed Description	55
8.26 Pardiso_data Struct Reference	55
8.26.1 Detailed Description	55
8.27 precondition Struct Reference	56
8.27.1 Detailed Description	56
8.28 precondition_data Struct Reference	56
8.28.1 Detailed Description	58
8.29 precondition_data_blc Struct Reference	58
8.29.1 Detailed Description	58
8.29.2 Field Documentation	59
8.29.2.1 A_diag	59
8.29.2.2 Ablc	59
8.29.2.3 amgparam	59

8.29.2.4 LU_diag . . . . .	59
8.29.2.5 mgl . . . . .	60
8.29.2.6 r . . . . .	60
8.30 precondition_data_bsr Struct Reference . . . . .	60
8.30.1 Detailed Description . . . . .	61
8.31 precondition_data_str Struct Reference . . . . .	62
8.31.1 Detailed Description . . . . .	63
8.32 precondition_data_sweeping Struct Reference . . . . .	63
8.32.1 Detailed Description . . . . .	64
8.32.2 Field Documentation . . . . .	64
8.32.2.1 A . . . . .	64
8.32.2.2 Ai . . . . .	65
8.32.2.3 local_A . . . . .	65
8.32.2.4 local_index . . . . .	65
8.32.2.5 local_LU . . . . .	65
8.32.2.6 NumLayers . . . . .	65
8.32.2.7 r . . . . .	66
8.32.2.8 w . . . . .	66
8.33 precondition_diag_bsr Struct Reference . . . . .	66
8.33.1 Detailed Description . . . . .	66
8.34 precondition_diag_str Struct Reference . . . . .	67
8.34.1 Detailed Description . . . . .	67
8.35 SWZ_data Struct Reference . . . . .	67
8.35.1 Detailed Description . . . . .	68
8.36 SWZ_param Struct Reference . . . . .	69
8.36.1 Detailed Description . . . . .	69
<b>9 File Documentation</b>	<b>71</b>
9.1 AuxArray.c File Reference . . . . .	71
9.1.1 Detailed Description . . . . .	71
9.1.2 Function Documentation . . . . .	72
9.1.2.1 fasp_darray_cp() . . . . .	72
9.1.2.2 fasp_darray_set() . . . . .	72
9.1.2.3 fasp_iarray_cp() . . . . .	73
9.1.2.4 fasp_iarray_set() . . . . .	73
9.2 AuxConvert.c File Reference . . . . .	74
9.2.1 Detailed Description . . . . .	74
9.2.2 Function Documentation . . . . .	74
9.2.2.1 fasp_aux_bbyteToldouble() . . . . .	74

---

9.2.2.2 fasp_aux_change_endian4()	75
9.2.2.3 fasp_aux_change_endian8()	75
9.3 AuxGivens.c File Reference	75
9.3.1 Detailed Description	76
9.3.2 Function Documentation	76
9.3.2.1 fasp_aux_givens()	76
9.4 AuxGraphics.c File Reference	76
9.4.1 Detailed Description	77
9.4.2 Function Documentation	77
9.4.2.1 fasp_dbsr_plot()	77
9.4.2.2 fasp_dbsr_subplot()	78
9.4.2.3 fasp_dcsr_plot()	78
9.4.2.4 fasp_dcsr_subplot()	79
9.4.2.5 fasp_grid2d_plot()	79
9.5 AuxInput.c File Reference	80
9.5.1 Detailed Description	80
9.5.2 Function Documentation	80
9.5.2.1 fasp_param_check()	80
9.5.2.2 fasp_param_input()	81
9.6 AuxMemory.c File Reference	81
9.6.1 Detailed Description	82
9.6.2 Function Documentation	82
9.6.2.1 fasp_mem_free()	82
9.6.2.2 fasp_mem_iludata_check()	82
9.6.2.3 fasp_mem_realloc()	83
9.6.2.4 fasp_mem_usage()	83
9.7 AuxMessage.c File Reference	84
9.7.1 Detailed Description	84
9.7.2 Function Documentation	84
9.7.2.1 fasp_amgcomplexity()	84
9.7.2.2 fasp_amgcomplexity_bsr()	85
9.7.2.3 fasp_chkerr()	85
9.7.2.4 fasp_cputime()	86
9.7.2.5 fasp_itinfo()	86
9.7.2.6 fasp_message()	87
9.8 AuxParam.c File Reference	87
9.8.1 Detailed Description	88
9.8.2 Function Documentation	88
9.8.2.1 fasp_param_amg_init()	88

---



9.8.2.2 fasp_param_amg_print()	89
9.8.2.3 fasp_param_amg_set()	89
9.8.2.4 fasp_param_amg_to_prec()	89
9.8.2.5 fasp_param_amg_to_precbsr()	91
9.8.2.6 fasp_param_ilu_init()	91
9.8.2.7 fasp_param_ilu_print()	92
9.8.2.8 fasp_param_ilu_set()	92
9.8.2.9 fasp_param_init()	92
9.8.2.10 fasp_param_input_init()	93
9.8.2.11 fasp_param_prec_to_amg()	93
9.8.2.12 fasp_param_precbsr_to_amg()	94
9.8.2.13 fasp_param_set()	94
9.8.2.14 fasp_param_solver_init()	95
9.8.2.15 fasp_param_solver_print()	95
9.8.2.16 fasp_param_solver_set()	95
9.8.2.17 fasp_param_swz_init()	96
9.8.2.18 fasp_param_swz_print()	96
9.8.2.19 fasp_param_swz_set()	97
9.9 AuxSort.c File Reference	97
9.9.1 Detailed Description	98
9.9.2 Function Documentation	98
9.9.2.1 fasp_aux_BiSearch()	98
9.9.2.2 fasp_aux_dQuickSort()	98
9.9.2.3 fasp_aux_dQuickSortIndex()	100
9.9.2.4 fasp_aux_iQuickSort()	101
9.9.2.5 fasp_aux_iQuickSortIndex()	101
9.9.2.6 fasp_aux_merge()	102
9.9.2.7 fasp_aux_msort()	102
9.9.2.8 fasp_aux_unique()	103
9.10 AuxThreads.c File Reference	103
9.10.1 Detailed Description	104
9.10.2 Function Documentation	104
9.10.2.1 fasp_get_start_end()	104
9.10.2.2 fasp_set_gs_threads()	105
9.10.3 Variable Documentation	105
9.10.3.1 THDs_AMG_GS	105
9.10.3.2 THDs_CPR_gGS	105
9.10.3.3 THDs_CPR_IGS	105
9.11 AuxTiming.c File Reference	105

9.11.1 Detailed Description	106
9.11.2 Function Documentation	106
9.11.2.1 fasp_gettime()	106
9.12 AuxVector.c File Reference	106
9.12.1 Detailed Description	107
9.12.2 Function Documentation	107
9.12.2.1 fasp_dvec_alloc()	107
9.12.2.2 fasp_dvec_cp()	108
9.12.2.3 fasp_dvec_create()	108
9.12.2.4 fasp_dvec_free()	109
9.12.2.5 fasp_dvec_isnan()	109
9.12.2.6 fasp_dvec_maxdiff()	109
9.12.2.7 fasp_dvec_rand()	110
9.12.2.8 fasp_dvec_set()	111
9.12.2.9 fasp_dvec_symdiagscale()	111
9.12.2.10 fasp_ivec_alloc()	112
9.12.2.11 fasp_ivec_create()	112
9.12.2.12 fasp_ivec_free()	113
9.12.2.13 fasp_ivec_set()	113
9.13 BlaArray.c File Reference	113
9.13.1 Detailed Description	114
9.13.2 Function Documentation	115
9.13.2.1 fasp_blas_darray_ax()	115
9.13.2.2 fasp_blas_darray_axpby()	115
9.13.2.3 fasp_blas_darray_axpy()	116
9.13.2.4 fasp_blas_darray_axpy_nc2()	116
9.13.2.5 fasp_blas_darray_axpy_nc3()	117
9.13.2.6 fasp_blas_darray_axpy_nc5()	117
9.13.2.7 fasp_blas_darray_axpy_nc7()	118
9.13.2.8 fasp_blas_darray_axpyz()	118
9.13.2.9 fasp_blas_darray_axpyz_nc2()	119
9.13.2.10 fasp_blas_darray_axpyz_nc3()	119
9.13.2.11 fasp_blas_darray_axpyz_nc5()	120
9.13.2.12 fasp_blas_darray_axpyz_nc7()	120
9.13.2.13 fasp_blas_darray_dotprod()	121
9.13.2.14 fasp_blas_darray_norm1()	121
9.13.2.15 fasp_blas_darray_norm2()	122
9.13.2.16 fasp_blas_darray_norminf()	122
9.14 BlaEigen.c File Reference	123

9.14.1 Detailed Description	123
9.14.2 Function Documentation	123
9.14.2.1 fasp_dcsr_maxeig()	123
9.15 BlaFormat.c File Reference	124
9.15.1 Detailed Description	124
9.15.2 Function Documentation	125
9.15.2.1 fasp_format_dblc_dcsr()	125
9.15.2.2 fasp_format_dbsr_dcoo()	125
9.15.2.3 fasp_format_dbsr_dcsr()	126
9.15.2.4 fasp_format_dcoo_dcsr()	126
9.15.2.5 fasp_format_dcsr_dbsr()	127
9.15.2.6 fasp_format_dcsr_dcoo()	127
9.15.2.7 fasp_format_dcsr_dcsr()	128
9.15.2.8 fasp_format_dstr_dbsr()	128
9.15.2.9 fasp_format_dstr_dcsr()	128
9.16 BlalLU.c File Reference	129
9.16.1 Detailed Description	129
9.16.2 Function Documentation	130
9.16.2.1 fasp_iluk()	130
9.16.2.2 fasp_ilut()	131
9.16.2.3 fasp_ilutp()	132
9.16.2.4 fasp_symbfactor()	133
9.17 BlalLUSetupBSR.c File Reference	135
9.17.1 Detailed Description	135
9.17.2 Function Documentation	136
9.17.2.1 fasp_ilu_dbsr_setup()	136
9.17.2.2 fasp_ilu_dbsr_setup_levsch_omp()	136
9.17.2.3 fasp_ilu_dbsr_setup_mc_omp()	137
9.17.2.4 fasp_ilu_dbsr_setup_omp()	138
9.18 BlalLUSetupCSR.c File Reference	138
9.18.1 Detailed Description	138
9.18.2 Function Documentation	139
9.18.2.1 fasp_ilu_dcsr_setup()	139
9.19 BlalLUSetupSTR.c File Reference	139
9.19.1 Detailed Description	139
9.19.2 Function Documentation	140
9.19.2.1 fasp_ilu_dstr_setup0()	140
9.19.2.2 fasp_ilu_dstr_setup1()	140
9.20 BlalO.c File Reference	141

---

9.20.1 Detailed Description	143
9.20.2 Function Documentation	143
9.20.2.1 fasp_dbsr_print()	143
9.20.2.2 fasp_dbsr_read()	143
9.20.2.3 fasp_dbsr_write()	144
9.20.2.4 fasp_dbsr_write_coo()	145
9.20.2.5 fasp_dcoo_print()	145
9.20.2.6 fasp_dcoo_read()	145
9.20.2.7 fasp_dcoo_read1()	146
9.20.2.8 fasp_dcoo_shift_read()	146
9.20.2.9 fasp_dcoo_write()	147
9.20.2.10 fasp_dcsr_print()	148
9.20.2.11 fasp_dcsr_read()	148
9.20.2.12 fasp_dcsr_write_coo()	148
9.20.2.13 fasp_dcsrvec_read1()	149
9.20.2.14 fasp_dcsrvec_read2()	149
9.20.2.15 fasp_dcsrvec_write1()	151
9.20.2.16 fasp_dcsrvec_write2()	152
9.20.2.17 fasp_dmtx_read()	153
9.20.2.18 fasp_dmtxsym_read()	153
9.20.2.19 fasp_dstr_print()	154
9.20.2.20 fasp_dstr_read()	154
9.20.2.21 fasp_dstr_write()	155
9.20.2.22 fasp_dvec_print()	155
9.20.2.23 fasp_dvec_read()	156
9.20.2.24 fasp_dvec_write()	156
9.20.2.25 fasp_dvecind_write()	157
9.20.2.26 fasp_hb_read()	157
9.20.2.27 fasp_ivec_print()	158
9.20.2.28 fasp_ivec_read()	158
9.20.2.29 fasp_ivec_write()	159
9.20.2.30 fasp_ivecind_read()	159
9.20.2.31 fasp_matrix_read()	160
9.20.2.32 fasp_matrix_read_bin()	161
9.20.2.33 fasp_matrix_write()	161
9.20.2.34 fasp_vector_read()	162
9.20.2.35 fasp_vector_write()	163
9.20.3 Variable Documentation	164
9.20.3.1 dlength	164

---

---

9.20.3.2 ilength . . . . .	164
9.21 BlaOrderingCSR.c File Reference . . . . .	164
9.21.1 Detailed Description . . . . .	164
9.21.2 Function Documentation . . . . .	164
9.21.2.1 fasp_dcsr_CMK_order() . . . . .	164
9.21.2.2 fasp_dcsr_RCMK_order() . . . . .	165
9.22 BlaSchwarzSetup.c File Reference . . . . .	165
9.22.1 Detailed Description . . . . .	166
9.22.2 Function Documentation . . . . .	166
9.22.2.1 fasp_dcsr_swz_backward() . . . . .	166
9.22.2.2 fasp_dcsr_swz_forward() . . . . .	166
9.22.2.3 fasp_swz_dcsr_setup() . . . . .	167
9.23 BlaSmallMat.c File Reference . . . . .	167
9.23.1 Detailed Description . . . . .	169
9.23.2 Function Documentation . . . . .	169
9.23.2.1 fasp_blas_smat_aAxpby() . . . . .	169
9.23.2.2 fasp_blas_smat_add() . . . . .	170
9.23.2.3 fasp_blas_smat_axm() . . . . .	170
9.23.2.4 fasp_blas_smat_mul() . . . . .	171
9.23.2.5 fasp_blas_smat_mul_nc2() . . . . .	171
9.23.2.6 fasp_blas_smat_mul_nc3() . . . . .	172
9.23.2.7 fasp_blas_smat_mul_nc4() . . . . .	172
9.23.2.8 fasp_blas_smat_mul_nc5() . . . . .	173
9.23.2.9 fasp_blas_smat_mul_nc7() . . . . .	173
9.23.2.10 fasp_blas_smat_m xv() . . . . .	174
9.23.2.11 fasp_blas_smat_m xv_nc2() . . . . .	174
9.23.2.12 fasp_blas_smat_m xv_nc3() . . . . .	175
9.23.2.13 fasp_blas_smat_m xv_nc4() . . . . .	175
9.23.2.14 fasp_blas_smat_m xv_nc5() . . . . .	176
9.23.2.15 fasp_blas_smat_m xv_nc7() . . . . .	176
9.23.2.16 fasp_blas_smat_ymAx() . . . . .	177
9.23.2.17 fasp_blas_smat_ymAx_nc2() . . . . .	177
9.23.2.18 fasp_blas_smat_ymAx_nc3() . . . . .	178
9.23.2.19 fasp_blas_smat_ymAx_nc4() . . . . .	178
9.23.2.20 fasp_blas_smat_ymAx_nc5() . . . . .	179
9.23.2.21 fasp_blas_smat_ymAx_nc7() . . . . .	179
9.23.2.22 fasp_blas_smat_ypAx() . . . . .	180
9.23.2.23 fasp_blas_smat_ypAx_nc2() . . . . .	180
9.23.2.24 fasp_blas_smat_ypAx_nc3() . . . . .	180

9.23.2.25 fasp_blas_smat_ypAx_nc4()	181
9.23.2.26 fasp_blas_smat_ypAx_nc5()	181
9.23.2.27 fasp_blas_smat_ypAx_nc7()	182
9.24 BlaSmallMatInv.c File Reference	182
9.24.1 Detailed Description	183
9.24.2 Macro Definition Documentation	183
9.24.2.1 SWAP	183
9.24.3 Function Documentation	184
9.24.3.1 fasp_smat_identity()	184
9.24.3.2 fasp_smat_identity_nc2()	184
9.24.3.3 fasp_smat_identity_nc3()	184
9.24.3.4 fasp_smat_identity_nc5()	185
9.24.3.5 fasp_smat_identity_nc7()	185
9.24.3.6 fasp_smat_inv()	186
9.24.3.7 fasp_smat_inv_nc()	186
9.24.3.8 fasp_smat_inv_nc2()	186
9.24.3.9 fasp_smat_inv_nc3()	187
9.24.3.10 fasp_smat_inv_nc4()	187
9.24.3.11 fasp_smat_inv_nc5()	187
9.24.3.12 fasp_smat_inv_nc7()	188
9.24.3.13 fasp_smat_invp_nc()	188
9.24.3.14 fasp_smat_Linf()	189
9.25 BlaSmallMatLU.c File Reference	189
9.25.1 Detailed Description	189
9.25.2 Function Documentation	190
9.25.2.1 fasp_smat_lu_decomp()	190
9.25.2.2 fasp_smat_lu_solve()	190
9.26 BlaSparseBLC.c File Reference	191
9.26.1 Detailed Description	191
9.26.2 Function Documentation	192
9.26.2.1 fasp_dblc_free()	192
9.27 BlaSparseBSR.c File Reference	192
9.27.1 Detailed Description	193
9.27.2 Function Documentation	193
9.27.2.1 fasp_dbsr_alloc()	193
9.27.2.2 fasp_dbsr_cp()	194
9.27.2.3 fasp_dbsr_create()	194
9.27.2.4 fasp_dbsr_diaginv()	195
9.27.2.5 fasp_dbsr_diaginv2()	195

9.27.2.6 fasp_dbsr_diaginv3()	196
9.27.2.7 fasp_dbsr_diaginv4()	197
9.27.2.8 fasp_dbsr_diagLU()	197
9.27.2.9 fasp_dbsr_diagLU2()	198
9.27.2.10 fasp_dbsr_diagpref()	198
9.27.2.11 fasp_dbsr_free()	199
9.27.2.12 fasp_dbsr_getblk()	199
9.27.2.13 fasp_dbsr_getdiag()	200
9.27.2.14 fasp_dbsr_getdiaginv()	200
9.27.2.15 fasp_dbsr_merge_col()	202
9.27.2.16 fasp_dbsr_perm()	202
9.27.2.17 fasp_dbsr_trans()	203
9.28 BlaSparseCheck.c File Reference	203
9.28.1 Detailed Description	204
9.28.2 Function Documentation	204
9.28.2.1 fasp_check_dCSRmat()	204
9.28.2.2 fasp_check_diagdom()	204
9.28.2.3 fasp_check_diagpos()	205
9.28.2.4 fasp_check_diagzero()	205
9.28.2.5 fasp_check_iCSRmat()	206
9.28.2.6 fasp_check_ordering()	206
9.28.2.7 fasp_check_symm()	207
9.29 BlaSparseCOO.c File Reference	207
9.29.1 Detailed Description	208
9.29.2 Function Documentation	208
9.29.2.1 fasp_dcoo_alloc()	208
9.29.2.2 fasp_dcoo_create()	208
9.29.2.3 fasp_dcoo_free()	209
9.29.2.4 fasp_dcoo_shift()	209
9.30 BlaSparseCSR.c File Reference	210
9.30.1 Detailed Description	211
9.30.2 Function Documentation	211
9.30.2.1 fasp_dcsr_alloc()	211
9.30.2.2 fasp_dcsr_bandwidth()	212
9.30.2.3 fasp_dcsr_compress()	212
9.30.2.4 fasp_dcsr_compress_inplace()	213
9.30.2.5 fasp_dcsr_cp()	213
9.30.2.6 fasp_dcsr_create()	214
9.30.2.7 fasp_dcsr_diagpref()	214

9.30.2.8 fasp_dcsr_free()	215
9.30.2.9 fasp_dcsr_getblk()	215
9.30.2.10 fasp_dcsr_getcol()	216
9.30.2.11 fasp_dcsr_getdiag()	216
9.30.2.12 fasp_dcsr_multicoloring()	218
9.30.2.13 fasp_dcsr_perm()	218
9.30.2.14 fasp_dcsr_permz()	219
9.30.2.15 fasp_dcsr_regdiag()	220
9.30.2.16 fasp_dcsr_shift()	221
9.30.2.17 fasp_dcsr_sort()	221
9.30.2.18 fasp_dcsr_sortz()	222
9.30.2.19 fasp_dcsr_symdiagscale()	222
9.30.2.20 fasp_dcsr_sympart()	223
9.30.2.21 fasp_dcsr_trans()	223
9.30.2.22 fasp_dcsr_transpose()	224
9.30.2.23 fasp_dcsr_transz()	224
9.30.2.24 fasp_icsr_cp()	225
9.30.2.25 fasp_icsr_create()	225
9.30.2.26 fasp_icsr_free()	226
9.30.2.27 fasp_icsr_trans()	226
9.31 BlaSparseCSRL.c File Reference	227
9.31.1 Detailed Description	227
9.31.2 Function Documentation	227
9.31.2.1 fasp_dcsrl_create()	227
9.31.2.2 fasp_dcsrl_free()	228
9.32 BlaSparseSTR.c File Reference	228
9.32.1 Detailed Description	229
9.32.2 Function Documentation	229
9.32.2.1 fasp_dstr_alloc()	229
9.32.2.2 fasp_dstr_cp()	230
9.32.2.3 fasp_dstr_create()	230
9.32.2.4 fasp_dstr_free()	231
9.33 BlaSparseUtil.c File Reference	231
9.33.1 Detailed Description	232
9.33.2 Function Documentation	232
9.33.2.1 fasp_sparse_aat_()	232
9.33.2.2 fasp_sparse_abyb_()	233
9.33.2.3 fasp_sparse_abybms_()	234
9.33.2.4 fasp_sparse_aplbms_()	234



9.33.2.5 fasp_sparse_aplusb_()	235
9.33.2.6 fasp_sparse_iit_()	235
9.33.2.7 fasp_sparse_mis_()	236
9.33.2.8 fasp_sparse_rapcmp_()	236
9.33.2.9 fasp_sparse_rapms_()	237
9.33.2.10 fasp_sparse_wta_()	238
9.33.2.11 fasp_sparse_wtams_()	239
9.33.2.12 fasp_sparse_ytx_()	239
9.33.2.13 fasp_sparse_ytxbig_()	240
9.34 BlaSpmvBLC.c File Reference	240
9.34.1 Detailed Description	241
9.34.2 Function Documentation	241
9.34.2.1 fasp_blas_dblc_aApy()	241
9.34.2.2 fasp_blas_dblc_mxv()	241
9.35 BlaSpmvBSR.c File Reference	242
9.35.1 Detailed Description	243
9.35.2 Function Documentation	243
9.35.2.1 fasp_blas_dbsr_aApy()	243
9.35.2.2 fasp_blas_dbsr_aApy()	243
9.35.2.3 fasp_blas_dbsr_aApy_agg()	245
9.35.2.4 fasp_blas_dbsr_axm()	246
9.35.2.5 fasp_blas_dbsr_mxm()	246
9.35.2.6 fasp_blas_dbsr_mxv()	246
9.35.2.7 fasp_blas_dbsr_mxv_agg()	247
9.35.2.8 fasp_blas_dbsr_rap()	248
9.35.2.9 fasp_blas_dbsr_rap1()	248
9.35.2.10 fasp_blas_dbsr_rap_agg()	249
9.36 BlaSpmvCSR.c File Reference	249
9.36.1 Detailed Description	250
9.36.2 Function Documentation	251
9.36.2.1 fasp_blas_dcsr_aApy()	251
9.36.2.2 fasp_blas_dcsr_aApy_agg()	251
9.36.2.3 fasp_blas_dcsr_add()	252
9.36.2.4 fasp_blas_dcsr_axm()	252
9.36.2.5 fasp_blas_dcsr_mxm()	253
9.36.2.6 fasp_blas_dcsr_mxv()	253
9.36.2.7 fasp_blas_dcsr_mxv_agg()	254
9.36.2.8 fasp_blas_dcsr_ptap()	254
9.36.2.9 fasp_blas_dcsr_rap()	255

9.36.2.10 fasp_blas_dcsr_rap2()	255
9.36.2.11 fasp_blas_dcsr_rap4()	256
9.36.2.12 fasp_blas_dcsr_rap_agg()	257
9.36.2.13 fasp_blas_dcsr_rap_agg1()	257
9.36.2.14 fasp_blas_dcsr_vmv()	258
9.37 BlaSpmvCSRL.c File Reference	258
9.37.1 Detailed Description	258
9.37.2 Function Documentation	259
9.37.2.1 fasp_blas_dcsr_mxv()	259
9.38 BlaSpmvSTR.c File Reference	259
9.38.1 Detailed Description	260
9.38.2 Function Documentation	260
9.38.2.1 fasp_blas_dstr_aApy()	260
9.38.2.2 fasp_blas_dstr_diagscale()	260
9.38.2.3 fasp_blas_dstr_mxv()	261
9.39 BlaVector.c File Reference	261
9.39.1 Detailed Description	262
9.39.2 Function Documentation	262
9.39.2.1 fasp_blas_dvec_axpy()	262
9.39.2.2 fasp_blas_dvec_axpyz()	262
9.39.2.3 fasp_blas_dvec_dotprod()	263
9.39.2.4 fasp_blas_dvec_norm1()	263
9.39.2.5 fasp_blas_dvec_norm2()	264
9.39.2.6 fasp_blas_dvec_norminf()	264
9.39.2.7 fasp_blas_dvec_reterr()	265
9.40 doxygen.h File Reference	265
9.40.1 Detailed Description	265
9.41 fasp.h File Reference	265
9.41.1 Detailed Description	268
9.41.2 Macro Definition Documentation	268
9.41.2.1 __FASP_HEADER__	268
9.41.2.2 ABS	268
9.41.2.3 DIAGONAL_PREF	268
9.41.2.4 DLMALLOC	269
9.41.2.5 FASP_VERSION	269
9.41.2.6 GE	269
9.41.2.7 GT	269
9.41.2.8 INT	269
9.41.2.9 ISNAN	269

9.41.2.10 LE	269
9.41.2.11 LONG	270
9.41.2.12 LONGLONG	270
9.41.2.13 LS	270
9.41.2.14 MAX	270
9.41.2.15 MIN	270
9.41.2.16 NEDMALLOC	270
9.41.2.17 PUT_INT	271
9.41.2.18 PUT_REAL	271
9.41.2.19 REAL	271
9.41.2.20 RS_C1	271
9.41.2.21 SHORT	271
9.41.2.22 STRLEN	271
9.41.3 Typedef Documentation	271
9.41.3.1 dCOOmat	272
9.41.3.2 dCSRLmat	272
9.41.3.3 dCSRmat	272
9.41.3.4 ddenmat	272
9.41.3.5 dSTRmat	272
9.41.3.6 dvector	272
9.41.3.7 iCOOmat	272
9.41.3.8 iCSRmat	272
9.41.3.9 idenmat	272
9.41.3.10 ivector	272
9.42 fasp_block.h File Reference	272
9.42.1 Detailed Description	273
9.42.2 Macro Definition Documentation	273
9.42.2.1 __FASPBLOCK_HEADER__	274
9.42.3 Typedef Documentation	274
9.42.3.1 block_dvector	274
9.42.3.2 block_ivector	274
9.42.3.3 dBLCmat	274
9.42.3.4 dBSRmat	274
9.42.3.5 iBLCmat	274
9.43 fasp_const.h File Reference	274
9.43.1 Detailed Description	278
9.43.2 Macro Definition Documentation	278
9.43.2.1 AMLI_CYCLE	278
9.43.2.2 ASCEND	278

9.43.2.3 BIGREAL	278
9.43.2.4 CF_ORDER	278
9.43.2.5 CGPT	279
9.43.2.6 CLASSIC_AMG	279
9.43.2.7 COARSE_AC	279
9.43.2.8 COARSE_CR	279
9.43.2.9 COARSE_MIS	279
9.43.2.10 COARSE_RS	279
9.43.2.11 COARSE_RSP	279
9.43.2.12 CPFIRST	279
9.43.2.13 DESCEND	280
9.43.2.14 ERROR_ALLOC_MEM	280
9.43.2.15 ERROR_AMG_COARSE_TYPE	280
9.43.2.16 ERROR_AMG_COARSEING	280
9.43.2.17 ERROR_AMG_INTERP_TYPE	280
9.43.2.18 ERROR_AMG_SETUP	280
9.43.2.19 ERROR_AMG_SMOOTH_TYPE	280
9.43.2.20 ERROR_DATA_STRUCTURE	280
9.43.2.21 ERROR_DATA_ZERODIAG	281
9.43.2.22 ERROR_DUMMY_VAR	281
9.43.2.23 ERROR_INPUT_PAR	281
9.43.2.24 ERROR_LIC_TYPE	281
9.43.2.25 ERROR_MAT_SIZE	281
9.43.2.26 ERROR_MISC	281
9.43.2.27 ERROR_NUM_BLOCKS	281
9.43.2.28 ERROR_OPEN_FILE	282
9.43.2.29 ERROR_QUAD_DIM	282
9.43.2.30 ERROR_QUAD_TYPE	282
9.43.2.31 ERROR_READ_FILE	282
9.43.2.32 ERROR_REGRESS	282
9.43.2.33 ERROR_SOLVER_EXIT	282
9.43.2.34 ERROR_SOLVER_ILUSETUP	282
9.43.2.35 ERROR_SOLVER_MAXIT	282
9.43.2.36 ERROR_SOLVER_MISC	283
9.43.2.37 ERROR_SOLVER_PRECTYPE	283
9.43.2.38 ERROR_SOLVER_SOLSTAG	283
9.43.2.39 ERROR_SOLVER_STAG	283
9.43.2.40 ERROR_SOLVER_TOLSMALL	283
9.43.2.41 ERROR_SOLVER_TYPE	283

---

9.43.2.42 ERROR_UNKNOWN . . . . .	283
9.43.2.43 ERROR_WRONG_FILE . . . . .	283
9.43.2.44 FALSE . . . . .	284
9.43.2.45 FASP_SUCCESS . . . . .	284
9.43.2.46 FGPT . . . . .	284
9.43.2.47 FPFIRST . . . . .	284
9.43.2.48 G0PT . . . . .	284
9.43.2.49 ILUK . . . . .	284
9.43.2.50 ILUt . . . . .	284
9.43.2.51 ILUtp . . . . .	285
9.43.2.52 INTERP_DIR . . . . .	285
9.43.2.53 INTERP_ENG . . . . .	285
9.43.2.54 INTERP_EXT . . . . .	285
9.43.2.55 INTERP_STD . . . . .	285
9.43.2.56 ISPT . . . . .	285
9.43.2.57 MAT_bBSR . . . . .	285
9.43.2.58 MAT_bCSR . . . . .	285
9.43.2.59 MAT_BLC . . . . .	286
9.43.2.60 MAT_BSR . . . . .	286
9.43.2.61 MAT_bSTR . . . . .	286
9.43.2.62 MAT_CSR . . . . .	286
9.43.2.63 MAT_CSRL . . . . .	286
9.43.2.64 MAT_FREE . . . . .	286
9.43.2.65 MAT_STR . . . . .	286
9.43.2.66 MAT_SymCSR . . . . .	286
9.43.2.67 MAX_AMG_LVL . . . . .	287
9.43.2.68 MAX_CRATE . . . . .	287
9.43.2.69 MAX_REFINE_LVL . . . . .	287
9.43.2.70 MAX_RESTART . . . . .	287
9.43.2.71 MAX_STAG . . . . .	287
9.43.2.72 MIN_CDOF . . . . .	287
9.43.2.73 MIN_CRATE . . . . .	287
9.43.2.74 NL_AMLI_CYCLE . . . . .	287
9.43.2.75 NO_ORDER . . . . .	288
9.43.2.76 NPAIR . . . . .	288
9.43.2.77 OFF . . . . .	288
9.43.2.78 ON . . . . .	288
9.43.2.79 OPENMP_HOLDS . . . . .	288
9.43.2.80 PAIRWISE . . . . .	288

9.43.2.81 PREC_AMG . . . . .	288
9.43.2.82 PREC_DIAG . . . . .	288
9.43.2.83 PREC_FMG . . . . .	289
9.43.2.84 PREC_ILU . . . . .	289
9.43.2.85 PREC_NULL . . . . .	289
9.43.2.86 PREC_SCHWARZ . . . . .	289
9.43.2.87 PRINT_ALL . . . . .	289
9.43.2.88 PRINT_MIN . . . . .	289
9.43.2.89 PRINT_MORE . . . . .	289
9.43.2.90 PRINT_MOST . . . . .	290
9.43.2.91 PRINT_NONE . . . . .	290
9.43.2.92 PRINT_SOME . . . . .	290
9.43.2.93 SA_AMG . . . . .	290
9.43.2.94 SCHWARZ_BACKWARD . . . . .	290
9.43.2.95 SCHWARZ_FORWARD . . . . .	290
9.43.2.96 SCHWARZ_SYMMETRIC . . . . .	290
9.43.2.97 SMALLREAL . . . . .	290
9.43.2.98 SMALLREAL2 . . . . .	291
9.43.2.99 SMOOTHER_BLKOIL . . . . .	291
9.43.2.100 SMOOTHER_CG . . . . .	291
9.43.2.101 SMOOTHER_GS . . . . .	291
9.43.2.102 SMOOTHER_GSOR . . . . .	291
9.43.2.103 SMOOTHER_JACOBI . . . . .	291
9.43.2.104 SMOOTHER_L1DIAG . . . . .	291
9.43.2.105 SMOOTHER_POLY . . . . .	292
9.43.2.106 SMOOTHER_SGS . . . . .	292
9.43.2.107 SMOOTHER_SGSOR . . . . .	292
9.43.2.108 SMOOTHER_SOR . . . . .	292
9.43.2.109 SMOOTHER_SPETEN . . . . .	292
9.43.2.110 SMOOTHER_SSOR . . . . .	292
9.43.2.111 SOLVER_AMG . . . . .	292
9.43.2.112 SOLVER_BiCGstab . . . . .	292
9.43.2.113 SOLVER_CG . . . . .	293
9.43.2.114 SOLVER_DEFAULT . . . . .	293
9.43.2.115 SOLVER_FMG . . . . .	293
9.43.2.116 SOLVER_GCG . . . . .	293
9.43.2.117 SOLVER_GCR . . . . .	293
9.43.2.118 SOLVER_GMRES . . . . .	293
9.43.2.119 SOLVER_MinRes . . . . .	293

9.43.2.120 SOLVER_MUMPS	293
9.43.2.121 SOLVER_PARDISO	294
9.43.2.122 SOLVER_SBiCGstab	294
9.43.2.123 SOLVER_SCG	294
9.43.2.124 SOLVER_SGCG	294
9.43.2.125 SOLVER_SGMRES	294
9.43.2.126 SOLVER_SMinRes	294
9.43.2.127 SOLVER_SUPERLU	294
9.43.2.128 SOLVER_SVFGMRES	294
9.43.2.129 SOLVER_SVGMRES	295
9.43.2.130 SOLVER_UMFPACK	295
9.43.2.131 SOLVER_VFGMRES	295
9.43.2.132 SOLVER_VGMRES	295
9.43.2.133 SPAIR	295
9.43.2.134 STAG_RATIO	295
9.43.2.135 STOP_MOD_REL_RES	295
9.43.2.136 STOP_REL_PRECRES	295
9.43.2.137 STOP_REL_RES	296
9.43.2.138 TRUE	296
9.43.2.139 UA_AMG	296
9.43.2.140 UNPT	296
9.43.2.141 USERDEFINED	296
9.43.2.142 V_CYCLE	296
9.43.2.143 VMB	296
9.43.2.144 VW_CYCLE	297
9.43.2.145 W_CYCLE	297
9.43.2.146 WV_CYCLE	297
9.44 fasp_grid.h File Reference	297
9.44.1 Detailed Description	297
9.44.2 Macro Definition Documentation	297
9.44.2.1 __FASPGRID_HEADER__	297
9.44.3 Typedef Documentation	298
9.44.3.1 grid2d	298
9.44.3.2 pcgrid2d	298
9.44.3.3 pgrid2d	298
9.45 ltrSmootherBSR.c File Reference	298
9.45.1 Detailed Description	299
9.45.2 Function Documentation	299
9.45.2.1 fasp_smoother_dbsr_gs()	299

9.45.2.2 fasp_smoother_dbsr_gs1()	300
9.45.2.3 fasp_smoother_dbsr_gs_ascend()	301
9.45.2.4 fasp_smoother_dbsr_gs_ascend1()	301
9.45.2.5 fasp_smoother_dbsr_gs_descend()	302
9.45.2.6 fasp_smoother_dbsr_gs_descend1()	302
9.45.2.7 fasp_smoother_dbsr_gs_order1()	303
9.45.2.8 fasp_smoother_dbsr_gs_order2()	303
9.45.2.9 fasp_smoother_dbsr_ilu()	304
9.45.2.10 fasp_smoother_dbsr_jacobi()	304
9.45.2.11 fasp_smoother_dbsr_jacobi1()	305
9.45.2.12 fasp_smoother_dbsr_jacobi_setup()	305
9.45.2.13 fasp_smoother_dbsr_sor()	306
9.45.2.14 fasp_smoother_dbsr_sor1()	306
9.45.2.15 fasp_smoother_dbsr_sor_ascend()	307
9.45.2.16 fasp_smoother_dbsr_sor_descend()	308
9.45.2.17 fasp_smoother_dbsr_sor_order()	308
9.45.3 Variable Documentation	309
9.45.3.1 ilu_solve_time	309
9.46 ltrSmootherCSR.c File Reference	309
9.46.1 Detailed Description	310
9.46.2 Function Documentation	310
9.46.2.1 fasp_smoother_dcsr_gs()	310
9.46.2.2 fasp_smoother_dcsr_gs_cf()	311
9.46.2.3 fasp_smoother_dcsr_ilu()	311
9.46.2.4 fasp_smoother_dcsr_jacobi()	312
9.46.2.5 fasp_smoother_dcsr_kaczmarz()	312
9.46.2.6 fasp_smoother_dcsr_L1diag()	313
9.46.2.7 fasp_smoother_dcsr_sgs()	314
9.46.2.8 fasp_smoother_dcsr_sor()	314
9.46.2.9 fasp_smoother_dcsr_sor_cf()	315
9.47 ltrSmootherCSRcr.c File Reference	316
9.47.1 Detailed Description	316
9.47.2 Function Documentation	316
9.47.2.1 fasp_smoother_dcsr_gscr()	317
9.48 ltrSmootherCSRpoly.c File Reference	317
9.48.1 Detailed Description	318
9.48.2 Function Documentation	318
9.48.2.1 fasp_smoother_dcsr_poly()	318
9.48.2.2 fasp_smoother_dcsr_poly_old()	319



9.49 ltrSmootherSTR.c File Reference	319
9.49.1 Detailed Description	320
9.49.2 Function Documentation	320
9.49.2.1 fasp_generate_diaginv_block()	321
9.49.2.2 fasp_smoother_dstr_gs()	321
9.49.2.3 fasp_smoother_dstr_gs1()	322
9.49.2.4 fasp_smoother_dstr_gs_ascend()	322
9.49.2.5 fasp_smoother_dstr_gs_cf()	323
9.49.2.6 fasp_smoother_dstr_gs_descend()	323
9.49.2.7 fasp_smoother_dstr_gs_order()	324
9.49.2.8 fasp_smoother_dstr_jacobi()	324
9.49.2.9 fasp_smoother_dstr_jacobi1()	325
9.49.2.10 fasp_smoother_dstr_sor()	325
9.49.2.11 fasp_smoother_dstr_sor1()	326
9.49.2.12 fasp_smoother_dstr_sor_ascend()	327
9.49.2.13 fasp_smoother_dstr_sor_cf()	327
9.49.2.14 fasp_smoother_dstr_sor_descend()	328
9.49.2.15 fasp_smoother_dstr_sor_order()	328
9.50 KryPbcgs.c File Reference	329
9.50.1 Detailed Description	330
9.50.2 Function Documentation	330
9.50.2.1 fasp_solver_dblc_pbcgs()	330
9.50.2.2 fasp_solver_dbsr_pbcgs()	331
9.50.2.3 fasp_solver_dcsr_pbcgs()	331
9.50.2.4 fasp_solver_dstr_pbcgs()	332
9.50.2.5 fasp_solver_pbcgs()	333
9.51 KryPcg.c File Reference	334
9.51.1 Detailed Description	334
9.51.2 Function Documentation	335
9.51.2.1 fasp_solver_dblc_pcg()	335
9.51.2.2 fasp_solver_dbsr_pcg()	336
9.51.2.3 fasp_solver_dcsr_pcg()	337
9.51.2.4 fasp_solver_dstr_pcg()	337
9.51.2.5 fasp_solver_pcg()	338
9.52 KryPgcg.c File Reference	339
9.52.1 Detailed Description	339
9.52.2 Function Documentation	339
9.52.2.1 fasp_solver_dcsr_pgcg()	340
9.52.2.2 fasp_solver_pgcg()	340

9.53 KryPgcr.c File Reference . . . . .	341
9.53.1 Detailed Description . . . . .	341
9.53.2 Function Documentation . . . . .	342
9.53.2.1 fasp_solver_dblc_pgcr() . . . . .	342
9.53.2.2 fasp_solver_dcsr_pgcr() . . . . .	342
9.54 KryPgmres.c File Reference . . . . .	343
9.54.1 Detailed Description . . . . .	344
9.54.2 Function Documentation . . . . .	344
9.54.2.1 fasp_solver_dblc_pgmres() . . . . .	344
9.54.2.2 fasp_solver_dbsr_pgmres() . . . . .	345
9.54.2.3 fasp_solver_dcsr_pgmres() . . . . .	346
9.54.2.4 fasp_solver_dstr_pgmres() . . . . .	347
9.54.2.5 fasp_solver_pgmres() . . . . .	347
9.55 KryPminres.c File Reference . . . . .	348
9.55.1 Detailed Description . . . . .	349
9.55.2 Function Documentation . . . . .	349
9.55.2.1 fasp_solver_dblc_pminres() . . . . .	349
9.55.2.2 fasp_solver_dcsr_pminres() . . . . .	350
9.55.2.3 fasp_solver_dstr_pminres() . . . . .	351
9.55.2.4 fasp_solver_pminres() . . . . .	351
9.56 KryPvfgmres.c File Reference . . . . .	352
9.56.1 Detailed Description . . . . .	353
9.56.2 Function Documentation . . . . .	353
9.56.2.1 fasp_solver_dblc_pvfgmres() . . . . .	353
9.56.2.2 fasp_solver_dbsr_pvfgmres() . . . . .	354
9.56.2.3 fasp_solver_dcsr_pvfgmres() . . . . .	355
9.56.2.4 fasp_solver_pvfgmres() . . . . .	356
9.57 KryPvgmres.c File Reference . . . . .	356
9.57.1 Detailed Description . . . . .	357
9.57.2 Function Documentation . . . . .	357
9.57.2.1 fasp_solver_dblc_pvgmres() . . . . .	357
9.57.2.2 fasp_solver_dbsr_pvgmres() . . . . .	358
9.57.2.3 fasp_solver_dcsr_pvgmres() . . . . .	359
9.57.2.4 fasp_solver_dstr_pvgmres() . . . . .	360
9.57.2.5 fasp_solver_pvgmres() . . . . .	360
9.58 KrySPbcgs.c File Reference . . . . .	361
9.58.1 Detailed Description . . . . .	362
9.58.2 Function Documentation . . . . .	362
9.58.2.1 fasp_solver_dblc_spbcs() . . . . .	362

9.58.2.2 fasp_solver_dbsr_spgcg()	363
9.58.2.3 fasp_solver_dcsr_spgcg()	364
9.58.2.4 fasp_solver_dstr_spgcg()	364
9.59 KrySPcg.c File Reference	365
9.59.1 Detailed Description	366
9.59.2 Function Documentation	366
9.59.2.1 fasp_solver_dblc_spcg()	366
9.59.2.2 fasp_solver_dcsr_spcg()	367
9.59.2.3 fasp_solver_dstr_spcg()	367
9.60 KrySPgmres.c File Reference	368
9.60.1 Detailed Description	369
9.60.2 Function Documentation	369
9.60.2.1 fasp_solver_dblc_spgmres()	369
9.60.2.2 fasp_solver_dbsr_spgmres()	370
9.60.2.3 fasp_solver_dcsr_spgmres()	370
9.60.2.4 fasp_solver_dstr_spgmres()	371
9.61 KrySPminres.c File Reference	372
9.61.1 Detailed Description	372
9.61.2 Function Documentation	373
9.61.2.1 fasp_solver_dblc_spmminres()	373
9.61.2.2 fasp_solver_dcsr_spmminres()	374
9.61.2.3 fasp_solver_dstr_spmminres()	374
9.62 KrySPvgmres.c File Reference	375
9.62.1 Detailed Description	376
9.62.2 Function Documentation	376
9.62.2.1 fasp_solver_dblc_spvgmres()	376
9.62.2.2 fasp_solver_dbsr_spvgmres()	377
9.62.2.3 fasp_solver_dcsr_spvgmres()	378
9.62.2.4 fasp_solver_dstr_spvgmres()	378
9.63 PreAMGCoarsenCR.c File Reference	379
9.63.1 Detailed Description	379
9.63.2 Function Documentation	380
9.63.2.1 fasp_amg_coarsening_cr()	380
9.64 PreAMGCoarsenRS.c File Reference	380
9.64.1 Detailed Description	381
9.64.2 Function Documentation	381
9.64.2.1 fasp_amg_coarsening_rs()	381
9.65 PreAMGInterp.c File Reference	382
9.65.1 Detailed Description	382

9.65.2 Function Documentation	382
9.65.2.1 fasp_amg_interp()	382
9.66 PreAMGInterpEM.c File Reference	383
9.66.1 Detailed Description	383
9.66.2 Function Documentation	383
9.66.2.1 fasp_amg_interp_em()	384
9.67 PreAMGSetupCR.c File Reference	384
9.67.1 Detailed Description	384
9.67.2 Function Documentation	385
9.67.2.1 fasp_amg_setup_cr()	385
9.68 PreAMGSetupRS.c File Reference	385
9.68.1 Detailed Description	385
9.68.2 Function Documentation	386
9.68.2.1 fasp_amg_setup_rs()	386
9.69 PreAMGSetupSA.c File Reference	386
9.69.1 Detailed Description	387
9.69.2 Function Documentation	387
9.69.2.1 fasp_amg_setup_sa()	387
9.70 PreAMGSetupSABSR.c File Reference	387
9.70.1 Detailed Description	388
9.70.2 Function Documentation	388
9.70.2.1 fasp_amg_setup_sa_bsr()	388
9.71 PreAMGSetupUA.c File Reference	389
9.71.1 Detailed Description	389
9.71.2 Function Documentation	389
9.71.2.1 fasp_amg_setup_ua()	389
9.72 PreAMGSetupUABSR.c File Reference	390
9.72.1 Detailed Description	390
9.72.2 Function Documentation	390
9.72.2.1 fasp_amg_setup_ua_bsr()	390
9.73 PreBLC.c File Reference	391
9.73.1 Detailed Description	392
9.73.2 Function Documentation	392
9.73.2.1 fasp_precond_dbic_diag_3()	392
9.73.2.2 fasp_precond_dbic_diag_3_amg()	392
9.73.2.3 fasp_precond_dbic_diag_4()	393
9.73.2.4 fasp_precond_dbic_lower_3()	393
9.73.2.5 fasp_precond_dbic_lower_3_amg()	395
9.73.2.6 fasp_precond_dbic_lower_4()	395

9.73.2.7 fasp_precond_dblc_SGS_3()	397
9.73.2.8 fasp_precond_dblc_SGS_3_amg()	397
9.73.2.9 fasp_precond_dblc_sweeping()	399
9.73.2.10 fasp_precond_dblc_upper_3()	399
9.73.2.11 fasp_precond_dblc_upper_3_amg()	401
9.74 PreBSR.c File Reference	401
9.74.1 Detailed Description	402
9.74.2 Function Documentation	402
9.74.2.1 fasp_precond_dbsr_amg()	402
9.74.2.2 fasp_precond_dbsr_amg_nk()	403
9.74.2.3 fasp_precond_dbsr_diag()	403
9.74.2.4 fasp_precond_dbsr_diag_nc2()	404
9.74.2.5 fasp_precond_dbsr_diag_nc3()	404
9.74.2.6 fasp_precond_dbsr_diag_nc5()	405
9.74.2.7 fasp_precond_dbsr_diag_nc7()	405
9.74.2.8 fasp_precond_dbsr_ilu()	406
9.74.2.9 fasp_precond_dbsr_ilu_ls_omp()	406
9.74.2.10 fasp_precond_dbsr_ilu_mc_omp()	407
9.74.2.11 fasp_precond_dbsr_namli()	408
9.75 PreCSR.c File Reference	408
9.75.1 Detailed Description	409
9.75.2 Function Documentation	409
9.75.2.1 fasp_precond_amg()	409
9.75.2.2 fasp_precond_amg_nk()	409
9.75.2.3 fasp_precond_amli()	410
9.75.2.4 fasp_precond_diag()	410
9.75.2.5 fasp_precond_famg()	411
9.75.2.6 fasp_precond_ilu()	411
9.75.2.7 fasp_precond_ilu_backward()	412
9.75.2.8 fasp_precond_ilu_forward()	412
9.75.2.9 fasp_precond_namli()	413
9.75.2.10 fasp_precond_setup()	413
9.75.2.11 fasp_precond_swz()	414
9.76 PreDataInit.c File Reference	414
9.76.1 Detailed Description	415
9.76.2 Function Documentation	415
9.76.2.1 fasp_amg_data_bsr_create()	415
9.76.2.2 fasp_amg_data_bsr_free()	416
9.76.2.3 fasp_amg_data_create()	416

9.76.2.4 fasp_amg_data_free()	416
9.76.2.5 fasp_ilu_data_create()	417
9.76.2.6 fasp_ilu_data_free()	417
9.76.2.7 fasp_precond_data_init()	418
9.76.2.8 fasp_swz_data_free()	418
9.77 PreMGCycle.c File Reference	418
9.77.1 Detailed Description	419
9.77.2 Function Documentation	419
9.77.2.1 fasp_solver_mgcycle()	419
9.77.2.2 fasp_solver_mgcycle_bsr()	420
9.78 PreMGCycleFull.c File Reference	420
9.78.1 Detailed Description	420
9.78.2 Function Documentation	420
9.78.2.1 fasp_solver_fmgcycle()	421
9.79 PreMGRecur.c File Reference	421
9.79.1 Detailed Description	421
9.79.2 Function Documentation	422
9.79.2.1 fasp_solver_mgrecur()	422
9.80 PreMGRecurAMLI.c File Reference	422
9.80.1 Detailed Description	423
9.80.2 Function Documentation	423
9.80.2.1 fasp_amg_amli_coef()	423
9.80.2.2 fasp_solver_amli()	423
9.80.2.3 fasp_solver_namli()	424
9.80.2.4 fasp_solver_namli_bsr()	425
9.81 PreMGSolve.c File Reference	425
9.81.1 Detailed Description	426
9.81.2 Function Documentation	426
9.81.2.1 fasp_amg_solve()	426
9.81.2.2 fasp_amg_solve_amli()	426
9.81.2.3 fasp_amg_solve_namli()	427
9.81.2.4 fasp_famg_solve()	428
9.82 PreSTR.c File Reference	428
9.82.1 Detailed Description	429
9.82.2 Function Documentation	429
9.82.2.1 fasp_precond_dstr_blockgs()	429
9.82.2.2 fasp_precond_dstr_diag()	429
9.82.2.3 fasp_precond_dstr_ilu0()	430
9.82.2.4 fasp_precond_dstr_ilu0_backward()	430

9.82.2.5 fasp_precond_dstr_ilu0_forward()	431
9.82.2.6 fasp_precond_dstr_ilu1()	431
9.82.2.7 fasp_precond_dstr_ilu1_backward()	432
9.82.2.8 fasp_precond_dstr_ilu1_forward()	432
9.83 SolAMG.c File Reference	432
9.83.1 Detailed Description	433
9.83.2 Function Documentation	433
9.83.2.1 fasp_solver_amg()	433
9.84 SolBLC.c File Reference	434
9.84.1 Detailed Description	434
9.84.2 Function Documentation	434
9.84.2.1 fasp_solver_dblc_itsolver()	435
9.84.2.2 fasp_solver_dblc_krylov()	435
9.84.2.3 fasp_solver_dblc_krylov_block3()	436
9.84.2.4 fasp_solver_dblc_krylov_block4()	437
9.84.2.5 fasp_solver_dblc_krylov_sweeping()	437
9.85 SolBSR.c File Reference	438
9.85.1 Detailed Description	439
9.85.2 Function Documentation	439
9.85.2.1 fasp_solver_dbsr_itsolver()	439
9.85.2.2 fasp_solver_dbsr_krylov()	439
9.85.2.3 fasp_solver_dbsr_krylov_amg()	440
9.85.2.4 fasp_solver_dbsr_krylov_amg_nk()	441
9.85.2.5 fasp_solver_dbsr_krylov_diag()	441
9.85.2.6 fasp_solver_dbsr_krylov_ilu()	442
9.85.2.7 fasp_solver_dbsr_krylov_nk_amg()	443
9.86 SolCSR.c File Reference	443
9.86.1 Detailed Description	444
9.86.2 Function Documentation	444
9.86.2.1 fasp_solver_dcsr_itsolver()	444
9.86.2.2 fasp_solver_dcsr_itsolver_s()	445
9.86.2.3 fasp_solver_dcsr_krylov()	446
9.86.2.4 fasp_solver_dcsr_krylov_amg()	446
9.86.2.5 fasp_solver_dcsr_krylov_amg_nk()	447
9.86.2.6 fasp_solver_dcsr_krylov_diag()	448
9.86.2.7 fasp_solver_dcsr_krylov_ilu()	448
9.86.2.8 fasp_solver_dcsr_krylov_ilu_M()	449
9.86.2.9 fasp_solver_dcsr_krylov_s()	449
9.86.2.10 fasp_solver_dcsr_krylov_swz()	450

9.87 SolFAMG.c File Reference	451
9.87.1 Detailed Description	451
9.87.2 Function Documentation	451
9.87.2.1 fasp_solver_famg()	451
9.88 SolGMGPoisson.c File Reference	452
9.88.1 Detailed Description	452
9.88.2 Function Documentation	452
9.88.2.1 fasp_poisson_fgmg1d()	453
9.88.2.2 fasp_poisson_fgmg2d()	453
9.88.2.3 fasp_poisson_fgmg3d()	454
9.88.2.4 fasp_poisson_gmg1d()	454
9.88.2.5 fasp_poisson_gmg2d()	456
9.88.2.6 fasp_poisson_gmg3d()	457
9.88.2.7 fasp_poisson_gmgcg1d()	457
9.88.2.8 fasp_poisson_gmgcg2d()	458
9.88.2.9 fasp_poisson_gmgcg3d()	459
9.89 SolMatFree.c File Reference	460
9.89.1 Detailed Description	460
9.89.2 Function Documentation	460
9.89.2.1 fasp_solver_itsolver()	460
9.89.2.2 fasp_solver_krylov()	461
9.89.2.3 fasp_solver_matfree_init()	462
9.90 SolSTR.c File Reference	462
9.90.1 Detailed Description	463
9.90.2 Function Documentation	463
9.90.2.1 fasp_solver_dstr_itsolver()	463
9.90.2.2 fasp_solver_dstr_krylov()	463
9.90.2.3 fasp_solver_dstr_krylov_blockgs()	464
9.90.2.4 fasp_solver_dstr_krylov_diag()	465
9.90.2.5 fasp_solver_dstr_krylov_ilu()	465
9.91 SolWrapper.c File Reference	466
9.91.1 Detailed Description	466
9.91.2 Function Documentation	467
9.91.2.1 fasp_fwrapper_dcsr_amg_()	467
9.91.2.2 fasp_fwrapper_dcsr_krylov_amg_()	468
9.91.2.3 fasp_fwrapper_dcsr_krylov_ilu_()	468
9.91.2.4 fasp_fwrapper_dcsr_pardiso_()	469
9.92 XtrMumps.c File Reference	470
9.92.1 Detailed Description	470



---

9.92.2 Macro Definition Documentation . . . . .	470
9.92.2.1 ICNTL . . . . .	470
9.92.3 Function Documentation . . . . .	470
9.92.3.1 fasp_solver_mumps() . . . . .	470
9.92.3.2 fasp_solver_mumps_steps() . . . . .	472
9.93 XtrPardiso.c File Reference . . . . .	472
9.93.1 Detailed Description . . . . .	473
9.93.2 Function Documentation . . . . .	473
9.93.2.1 fasp_solver_pardiso() . . . . .	473
9.94 XtrSamg.c File Reference . . . . .	473
9.94.1 Detailed Description . . . . .	474
9.94.2 Function Documentation . . . . .	474
9.94.2.1 dCSRmat2SAMGInput() . . . . .	474
9.94.2.2 dvector2SAMGInput() . . . . .	474
9.95 XtrSuperlu.c File Reference . . . . .	475
9.95.1 Detailed Description . . . . .	475
9.95.2 Function Documentation . . . . .	475
9.95.2.1 fasp_solver_superlu() . . . . .	475
9.96 XtrUmfpack.c File Reference . . . . .	476
9.96.1 Detailed Description . . . . .	476
9.96.2 Function Documentation . . . . .	476
9.96.2.1 fasp_solver_umfpack() . . . . .	476
<b>Index</b>	<b>479</b>



# Chapter 1

## Introduction

Over the last few decades, researchers have expended significant effort on developing efficient iterative methods for solving discretized partial differential equations (PDEs). Though these efforts have yielded many mathematically optimal solvers such as the multigrid method, the unfortunate reality is that multigrid methods have not been much used in practical applications. This marked gap between theory and practice is mainly due to the fragility of traditional multigrid (MG) methodology and the complexity of its implementation. We aim to develop techniques and the corresponding software that will narrow this gap, specifically by developing mathematically optimal solvers that are robust and easy to use in practice.

We believe that there is no one-size-for-all solution method for discrete linear systems from different applications. And, efficient iterative solvers can be constructed by taking the properties of PDEs and discretizations into account. In this project, we plan to construct a pool of discrete problems arising from partial differential equations (PDEs) or PDE systems and efficient linear solvers for these problems. We mainly utilize the methodology of Auxiliary Space Preconditioning (ASP) to construct efficient linear solvers. Due to this reason, this software package is called Fast Auxiliary Space Preconditioning or FASP for short.

The levels of abstraction are designed as follows:

- Level 0 (Aux\*.c): Auxiliary functions (timing, memory, threading, ...)
- Level 1 (Bla\*.c): Basic linear algebra subroutines (SpMV, RAP, ILU, SWZ, ...)
- Level 2 (Itr\*.c): Iterative methods and smoothers (Jacobi, GS, SOR, Poly, ...)
- Level 3 (Kry\*.c): Krylov iterative methods (CG, BiCGstab, MinRes, GMRES, ...)
- Level 4 (Pre\*.c): Preconditioners (GMG, AMG, FAMG, ...)
- Level 5 (Sol\*.c): User interface for FASP solvers (Solvers, wrappers, ...)
- Level x (Xtr\*.c): Interface to external packages (Mumps, Umfpack, ...)

FASP contains the kernel part and several applications (ranging from fluid dynamics to reservoir simulation). The kernel part is open-source and licensed under GNU Lesser General Public License or LGPL version 3.0 or later. Some of the applications contain contributions from and owned partially by other parties.

For the moment, FASP is under alpha testing. If you wish to obtain a current version of FASP or you have any questions, feel free to contact us at [faspdev@gmail.com](mailto:faspdev@gmail.com).

This software distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.



## Chapter 2

# How to obtain FASP

The most updated version of FASP can be downloaded from

<http://www.multigrid.org/fasp/download/faspsolver.zip>

We use HG (Mercurial) as our main version control tool. HG is easy to use and it is available at all OS platforms. For people who is interested in the developer version, you can obtain the FASP package with hg:

```
$ hg clone https://faspusers@bitbucket.org/fasp/faspsolver
```

will give you the developer version of the FASP package.



## Chapter 3

# Building and Installation

This is a simple instruction on building and testing. For more details, please refer to the README files and the short [User's Guide](#) in "faspolver/doc/".

To compile, you need a Fortran and a C compiler. First, you can type in the "faspolver/" root directory:

```
$ mkdir Build; cd Build; cmake ..
```

which will config the environment automatically. And, then, you can need to type:

```
$ make install
```

which will make the FASP shared static library and install to PREFIX/. By default, FASP libraries and executables will be installed in the FASP home directory "faspolver/".

There is a simple GUI tool for building and installing FASP included in the package. You need Tcl/Tk support in your computer. You may call this GUI by run in the root directory:

```
$ wish fasp_install.tcl
```

If you need to see the detailed usage of "make" or need any help, please type:

```
$ make help
```

After installation, tutorial examples can be found in "tutorial/".





## Chapter 4

# Developers

Project leader:

- Xu, Jinchao (Penn State University, USA)

Project coordinator:

- Zhang, Chensong (Chinese Academy of Sciences, China)

Current active developers (in alphabetic order):

- Feng, Chunsheng (Xiangtan University, China)
- Zhang, Chensong (Chinese Academy of Sciences, China)

With contributions from (in alphabetic order):

- Brannick, James (Penn State University, USA)
- Chen, Long (University of California, Irvine, USA)
- Hu, Xiaozhe (Tufts University, USA)
- Huang, Feiteng (Sichuang University, China)
- Huang, Xuehai (Shanghai Jiaotong University, China)
- Li, Zheng (Kunming University of Science and Technology, China)
- Qiao, Changhe (Penn State University, USA)
- Shu, Shi (Xiangtan University, China)
- Sun, Pengtao (University of Nevada, Las Vegas, USA)
- Yang, Kai (Penn State University, USA)

- Yue, Xiaoqiang (Xiangtan University, China)
- Wang, Lu (LLNL, USA)
- Wang, Ziteng (University of Alabama, USA)
- Zhang, Shiquan (Sichuan University, China)
- Zhang, Shuo (Chinese Academy of Sciences, China)
- Zhang, Hongxuan (Penn State Univeristy, USA)
- Zhang, Weifeng (Kunming University of Science and Technology, China)
- Zhou, Zhiyang (Xiangtan University, China)

## Chapter 5

# Doxygen

We use Doxygen as our automatically documentation generator which will make our future maintainance minimized. You can obtain the software (Windows, Linux and OS X) as well as its manual on the official website

<http://www.doxygen.org>

For an ordinary user, Doxygen is completely trivial to use. We only need to use some special marker in the usual comment as we put in c-files.



## Chapter 6

# Data Structure Index

### 6.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">AMG_data</a>	Data for AMG methods . . . . .	17
<a href="#">AMG_data_bsr</a>	Data for multigrid levels in <a href="#">dBSRmat</a> format . . . . .	18
<a href="#">AMG_param</a>	Parameters for AMG methods . . . . .	20
<a href="#">block_dvector</a>	Block REAL vector structure . . . . .	23
<a href="#">block_ivector</a>	Block INT vector structure . . . . .	23
<a href="#">dBLCmat</a>	Block REAL CSR matrix format . . . . .	24
<a href="#">dBSRmat</a>	Block sparse row storage matrix of REAL type . . . . .	24
<a href="#">dCOOmat</a>	Sparse matrix of REAL type in COO (IJ) format . . . . .	26
<a href="#">dCSRLmat</a>	Sparse matrix of REAL type in CSRL format . . . . .	26
<a href="#">dCSRmat</a>	Sparse matrix of REAL type in CSR format . . . . .	27
<a href="#">ddenmat</a>	Dense matrix of REAL type . . . . .	28
<a href="#">dSTRmat</a>	Structure matrix of REAL type . . . . .	29
<a href="#">dvector</a>	Vector with n entries of REAL type . . . . .	30
<a href="#">grid2d</a>	Two dimensional grid data structure . . . . .	30
<a href="#">iBLCmat</a>	Block INT CSR matrix format . . . . .	33
<a href="#">iCOOmat</a>	Sparse matrix of INT type in COO (IJ) format . . . . .	34

<a href="#">iCSRmat</a>	Sparse matrix of INT type in CSR format . . . . .	35
<a href="#">idenmat</a>	Dense matrix of INT type . . . . .	36
<a href="#">ILU_data</a>	Data for ILU setup . . . . .	36
<a href="#">ILU_param</a>	Parameters for ILU . . . . .	38
<a href="#">input_param</a>	Input parameters . . . . .	39
<a href="#">ITS_param</a>	Parameters for iterative solvers . . . . .	51
<a href="#">ivector</a>	Vector with n entries of INT type . . . . .	53
<a href="#">Mumps_data</a>	Data for MUMPS interface . . . . .	54
<a href="#">mxv_matfree</a>	Matrix-vector multiplication, replace the actual matrix . . . . .	55
<a href="#">Pardiso_data</a>	Data for Intel MKL PARDISO interface . . . . .	55
<a href="#">precond</a>	Preconditioner data and action . . . . .	56
<a href="#">precond_data</a>	Data for preconditioners . . . . .	56
<a href="#">precond_data_blc</a>	Data for block preconditioners in <a href="#">dBLCmat</a> format . . . . .	58
<a href="#">precond_data_bsr</a>	Data for preconditioners in <a href="#">dBSRmat</a> format . . . . .	60
<a href="#">precond_data_str</a>	Data for preconditioners in <a href="#">dSTRmat</a> format . . . . .	62
<a href="#">precond_data_sweeping</a>	Data for sweeping preconditioner . . . . .	63
<a href="#">precond_diag_bsr</a>	Data for diagonal preconditioners in <a href="#">dBSRmat</a> format . . . . .	66
<a href="#">precond_diag_str</a>	Data for diagonal preconditioners in <a href="#">dSTRmat</a> format . . . . .	67
<a href="#">SWZ_data</a>	Data for Schwarz methods . . . . .	67
<a href="#">SWZ_param</a>	Parameters for Schwarz method . . . . .	69

## Chapter 7

# File Index

### 7.1 File List

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

<a href="#">AuxArray.c</a>	Simple array operations – init, set, copy, etc . . . . .	71
<a href="#">AuxConvert.c</a>	Utilities for encoding format conversion . . . . .	74
<a href="#">AuxGivens.c</a>	Givens transformation . . . . .	75
<a href="#">AuxGraphics.c</a>	Graphical output for CSR matrix . . . . .	76
<a href="#">AuxInput.c</a>	Read and check input parameters . . . . .	80
<a href="#">AuxMemory.c</a>	Memory allocation and deallocation subroutines . . . . .	81
<a href="#">AuxMessage.c</a>	Output some useful messages . . . . .	84
<a href="#">AuxParam.c</a>	Initialize, set, or print input data and parameters . . . . .	87
<a href="#">AuxSort.c</a>	Array sorting/merging and removing duplicated integers . . . . .	97
<a href="#">AuxThreads.c</a>	Get and set number of threads and assign work load for each thread . . . . .	103
<a href="#">AuxTiming.c</a>	Timing subroutines . . . . .	105
<a href="#">AuxVector.c</a>	Simple vector operations – init, set, copy, etc . . . . .	106
<a href="#">BlaArray.c</a>	BLAS1 operations for arrays . . . . .	113
<a href="#">BlaEigen.c</a>	Computing the extreme eigenvalues . . . . .	123
<a href="#">BlaFormat.c</a>	Subroutines for matrix format conversion . . . . .	124
<a href="#">BlaLU.c</a>	Incomplete LU decomposition: ILUk, ILUt, ILUtp . . . . .	129

<a href="#">BlaLUSetupBSR.c</a>	Setup incomplete LU decomposition for <a href="#">dBSRmat</a> matrices . . . . .	135
<a href="#">BlaLUSetupCSR.c</a>	Setup incomplete LU decomposition for <a href="#">dCSRmat</a> matrices . . . . .	138
<a href="#">BlaLUSetupSTR.c</a>	Setup incomplete LU decomposition for <a href="#">dSTRmat</a> matrices . . . . .	139
<a href="#">BlaIO.c</a>	Matrix/vector input/output subroutines . . . . .	141
<a href="#">BlaOrderingCSR.c</a>	Generating ordering using algebraic information . . . . .	164
<a href="#">BlaSchwarzSetup.c</a>	Setup phase for the Schwarz methods . . . . .	165
<a href="#">BlaSmallMat.c</a>	BLAS operations for <i>small</i> dense matrices . . . . .	167
<a href="#">BlaSmallMatInv.c</a>	Find inversion of <i>small</i> dense matrices in row-major format . . . . .	182
<a href="#">BlaSmallMatLU.c</a>	LU decomposition and direct solver for small dense matrices . . . . .	189
<a href="#">BlaSparseBLC.c</a>	Sparse matrix block operations . . . . .	191
<a href="#">BlaSparseBSR.c</a>	Sparse matrix operations for <a href="#">dBSRmat</a> matrices . . . . .	192
<a href="#">BlaSparseCheck.c</a>	Check properties of sparse matrices . . . . .	203
<a href="#">BlaSparseCOO.c</a>	Sparse matrix operations for <a href="#">dCOOmat</a> matrices . . . . .	207
<a href="#">BlaSparseCSR.c</a>	Sparse matrix operations for <a href="#">dCSRmat</a> matrices . . . . .	210
<a href="#">BlaSparseCSRL.c</a>	Sparse matrix operations for <a href="#">dCSRLmat</a> matrices . . . . .	227
<a href="#">BlaSparseSTR.c</a>	Sparse matrix operations for <a href="#">dSTRmat</a> matrices . . . . .	228
<a href="#">BlaSparseUtil.c</a>	Routines for sparse matrix operations . . . . .	231
<a href="#">BlaSpmvBLC.c</a>	Linear algebraic operations for <a href="#">dBLCmat</a> matrices . . . . .	240
<a href="#">BlaSpmvBSR.c</a>	Linear algebraic operations for <a href="#">dBSRmat</a> matrices . . . . .	242
<a href="#">BlaSpmvCSR.c</a>	Linear algebraic operations for <a href="#">dCSRmat</a> matrices . . . . .	249
<a href="#">BlaSpmvCSRL.c</a>	Linear algebraic operations for <a href="#">dCSRLmat</a> matrices . . . . .	258
<a href="#">BlaSpmvSTR.c</a>	Linear algebraic operations for <a href="#">dSTRmat</a> matrices . . . . .	259
<a href="#">BlaVector.c</a>	BLAS1 operations for vectors . . . . .	261
<a href="#">doxygen.h</a>	Main page for Doygen documentation . . . . .	265
<a href="#">fasp.h</a>	Main header file for the FASP project . . . . .	265
<a href="#">fasp_block.h</a>	Header file for FASP block matrices . . . . .	272
<a href="#">fasp_const.h</a>	Definition of FASP constants, including messages, solver types, etc . . . . .	274



<a href="#">fasp_grid.h</a>	
Header file for FASP grid . . . . .	297
<a href="#">ltrSmootherBSR.c</a>	
Smoothers for <a href="#">dBSRmat</a> matrices . . . . .	298
<a href="#">ltrSmootherCSR.c</a>	
Smoothers for <a href="#">dCSRmat</a> matrices . . . . .	309
<a href="#">ltrSmootherCSRcr.c</a>	
Smoothers for <a href="#">dCSRmat</a> matrices using compatible relaxation . . . . .	316
<a href="#">ltrSmootherCSRpoly.c</a>	
Smoothers for <a href="#">dCSRmat</a> matrices using poly. approx. to $A^{-1}$ . . . . .	317
<a href="#">ltrSmootherSTR.c</a>	
Smoothers for <a href="#">dSTRmat</a> matrices . . . . .	319
<a href="#">KryPbcgs.c</a>	
Krylov subspace methods – Preconditioned BiCGstab . . . . .	329
<a href="#">KryPcg.c</a>	
Krylov subspace methods – Preconditioned CG . . . . .	334
<a href="#">KryPgcg.c</a>	
Krylov subspace methods – Preconditioned generalized CG . . . . .	339
<a href="#">KryPgcr.c</a>	
Krylov subspace methods – Preconditioned GCR . . . . .	341
<a href="#">KryPgmres.c</a>	
Krylov subspace methods – Right-preconditioned GMRes . . . . .	343
<a href="#">KryPminres.c</a>	
Krylov subspace methods – Preconditioned minimal residual . . . . .	348
<a href="#">KryPvfgmres.c</a>	
Krylov subspace methods – Preconditioned variable-restarting FGMRes . . . . .	352
<a href="#">KryPvgmres.c</a>	
Krylov subspace methods – Preconditioned variable-restart GMRes . . . . .	356
<a href="#">KrySPbcgs.c</a>	
Krylov subspace methods – Preconditioned BiCGstab with safety net . . . . .	361
<a href="#">KrySPcg.c</a>	
Krylov subspace methods – Preconditioned CG with safety net . . . . .	365
<a href="#">KrySPgmres.c</a>	
Krylov subspace methods – Preconditioned GMRes with safety net . . . . .	368
<a href="#">KrySPminres.c</a>	
Krylov subspace methods – Preconditioned MINRES with safety net . . . . .	372
<a href="#">KrySPvgmres.c</a>	
Krylov subspace methods – Preconditioned variable-restart GMRes with safety net . . . . .	375
<a href="#">PreAMGCoarsenCR.c</a>	
Coarsening with Brannick-Falgout strategy . . . . .	379
<a href="#">PreAMGCoarsenRS.c</a>	
Coarsening with a modified Ruge-Stuben strategy . . . . .	380
<a href="#">PreAMGInterp.c</a>	
Direct and standard interpolations for classical AMG . . . . .	382
<a href="#">PreAMGInterpEM.c</a>	
Interpolation operators for AMG based on energy-min . . . . .	383
<a href="#">PreAMGSetupCR.c</a>	
Brannick-Falgout compatible relaxation based AMG: SETUP phase . . . . .	384
<a href="#">PreAMGSetupRS.c</a>	
Ruge-Stuben AMG: SETUP phase . . . . .	385
<a href="#">PreAMGSetupSA.c</a>	
Smoothed aggregation AMG: SETUP phase . . . . .	386
<a href="#">PreAMGSetupSABSR.c</a>	
Smoothed aggregation AMG: SETUP phase (for BSR matrices) . . . . .	387

<a href="#">PreAMGSetupUA.c</a>	
Unsmoothed aggregation AMG: SETUP phase	389
<a href="#">PreAMGSetupUABSR.c</a>	
Unsmoothed aggregation AMG: SETUP phase (for BSR matrices)	390
<a href="#">PreBLC.c</a>	
Preconditioners for <a href="#">dBLCmat</a> matrices	391
<a href="#">PreBSR.c</a>	
Preconditioners for <a href="#">dBSRmat</a> matrices	401
<a href="#">PreCSR.c</a>	
Preconditioners for <a href="#">dCSRmat</a> matrices	408
<a href="#">PreDataInit.c</a>	
Initialize important data structures	414
<a href="#">PreMGCycle.c</a>	
Abstract multigrid cycle – non-recursive version	418
<a href="#">PreMGCycleFull.c</a>	
Abstract non-recursive full multigrid cycle	420
<a href="#">PreMGRecur.c</a>	
Abstract multigrid cycle – recursive version	421
<a href="#">PreMGRecurAMLI.c</a>	
Abstract AMLI multilevel iteration – recursive version	422
<a href="#">PreMGsSolve.c</a>	
Algebraic multigrid iterations: SOLVE phase	425
<a href="#">PreSTR.c</a>	
Preconditioners for <a href="#">dSTRmat</a> matrices	428
<a href="#">SolAMG.c</a>	
AMG method as an iterative solver	432
<a href="#">SolBLC.c</a>	
Iterative solvers for <a href="#">dBLCmat</a> matrices	434
<a href="#">SolBSR.c</a>	
Iterative solvers for <a href="#">dBSRmat</a> matrices	438
<a href="#">SolCSR.c</a>	
Iterative solvers for <a href="#">dCSRmat</a> matrices	443
<a href="#">SolFAMG.c</a>	
Full AMG method as an iterative solver	451
<a href="#">SolGMGPoisson.c</a>	
GMG method as an iterative solver for Poisson Problem	452
<a href="#">SolMatFree.c</a>	
Iterative solvers using MatFree spmv operations	460
<a href="#">SolISTR.c</a>	
Iterative solvers for <a href="#">dSTRmat</a> matrices	462
<a href="#">SolWrapper.c</a>	
Wrappers for accessing functions by advanced users	466
<a href="#">XtrMumps.c</a>	
Interface to MUMPS direct solvers	470
<a href="#">XtrPardiso.c</a>	
Interface to Intel MKL PARDISO direct solvers	472
<a href="#">XtrSamg.c</a>	
Interface to SAMG solvers	473
<a href="#">XtrSuperlu.c</a>	
Interface to SuperLU direct solvers	475
<a href="#">XtrUmfpack.c</a>	
Interface to UMFPACK direct solvers	476

## Chapter 8

# Data Structure Documentation

### 8.1 AMG\_data Struct Reference

Data for AMG methods.

```
#include <fasp.h>
```

#### Data Fields

- [SHORT max\\_levels](#)  
*max number of levels*
- [SHORT num\\_levels](#)  
*number of levels in use  $\leq$  max\_levels*
- [dCSRmat A](#)  
*pointer to the matrix at level level\_num*
- [dCSRmat R](#)  
*restriction operator at level level\_num*
- [dCSRmat P](#)  
*prolongation operator at level level\_num*
- [dvector b](#)  
*pointer to the right-hand side at level level\_num*
- [dvector x](#)  
*pointer to the iterative solution at level level\_num*
- `void *` [Numeric](#)  
*pointer to the numerical factorization from UMFPACK*
- [Pardiso\\_data pdata](#)  
*data for Intel MKL PARDISO*
- [ivector cfmark](#)  
*pointer to the CF marker at level level\_num*
- [INT ILU\\_levels](#)  
*number of levels use ILU smoother*

- [ILU\\_data LU](#)  
*ILU matrix for ILU smoother.*
- [INT near\\_kernel\\_dim](#)  
*dimension of the near kernel for SAMG*
- [REAL \\*\\* near\\_kernel\\_basis](#)  
*basis of near kernel space for SAMG*
- [INT SWZ\\_levels](#)  
*number of levels use Schwarz smoother*
- [SWZ\\_data Schwarz](#)  
*data of Schwarz smoother*
- [dvector w](#)  
*temporary work space*
- [Mumps\\_data mumps](#)  
*data for MUMPS*
- [INT cycle\\_type](#)  
*cycle type*
- [INT \\* ic](#)  
*indices for different colors*
- [INT \\* icmap](#)  
*mapping from vertex to color*
- [INT colors](#)  
*number of colors*
- [REAL weight](#)  
*weight for smoother*

### 8.1.1 Detailed Description

Data for AMG methods.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 777 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.2 AMG\_data\_bsr Struct Reference

Data for multigrid levels in [dBSRmat](#) format.

```
#include <fasp_block.h>
```

## Data Fields

- [INT max\\_levels](#)  
*max number of levels*
- [INT num\\_levels](#)  
*number of levels in use  $\leq$  max\_levels*
- [dBSRmat A](#)  
*pointer to the matrix at level level\_num*
- [dBSRmat R](#)  
*restriction operator at level level\_num*
- [dBSRmat P](#)  
*prolongation operator at level level\_num*
- [dvector b](#)  
*pointer to the right-hand side at level level\_num*
- [dvector x](#)  
*pointer to the iterative solution at level level\_num*
- [dvector diaginv](#)  
*pointer to the diagonal inverse at level level\_num*
- [dCSRmat Ac](#)  
*pointer to the matrix at level level\_num (csr format)*
- `void *` [Numeric](#)  
*pointer to the numerical factorization from UMFPACK*
- [Pardiso\\_data pdata](#)  
*data for Intel MKL PARDISO*
- [dCSRmat PP](#)  
*pointer to the pressure block (only for reservoir simulation)*
- `REAL *` [pw](#)  
*pointer to the auxiliary vectors for pressure block*
- [dBSRmat SS](#)  
*pointer to the saturation block (only for reservoir simulation)*
- `REAL *` [sw](#)  
*pointer to the auxiliary vectors for saturation block*
- [dvector diaginv\\_SS](#)  
*pointer to the diagonal inverse of the saturation block at level level\_num*
- [ILU\\_data PP\\_LU](#)  
*ILU data for pressure block.*
- [ivector cfmark](#)  
*pointer to the CF marker at level level\_num*
- [INT ILU\\_levels](#)  
*number of levels use ILU smoother*
- [ILU\\_data LU](#)  
*ILU matrix for ILU smoother.*
- [INT near\\_kernel\\_dim](#)  
*dimension of the near kernel for SAMG*
- `REAL **` [near\\_kernel\\_basis](#)  
*basis of near kernel space for SAMG*
- `dCSRmat *` [A\\_nk](#)

- [dCSRmat \\* P\\_nk](#)  
*Matrix data for near kernal.*
- [dCSRmat \\* R\\_nk](#)  
*Prolongation for near kernal.*
- [dvector w](#)  
*Resriction for near kernal.*
- [Mumps\\_data mumps](#)  
*temporary work space*  
*data for MUMPS*

### 8.2.1 Detailed Description

Data for multigrid levels in [dBSRmat](#) format.

#### Note

This structure is needed for the AMG solver/preconditioner in BSR format

Definition at line 146 of file fasp\_block.h.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.3 AMG\_param Struct Reference

Parameters for AMG methods.

```
#include <fasp.h>
```

### Data Fields

- [SHORT AMG\\_type](#)  
*type of AMG method*
- [SHORT print\\_level](#)  
*print level for AMG*
- [INT maxit](#)  
*max number of iterations of AMG*
- [REAL tol](#)  
*stopping tolerance for AMG solver*
- [SHORT max\\_levels](#)  
*max number of levels of AMG*
- [INT coarse\\_dof](#)  
*max number of coarsest level DOF*

- [SHORT cycle\\_type](#)  
*type of AMG cycle*
- [REAL quality\\_bound](#)  
*quality threshold for pairwise aggregation*
- [SHORT smoother](#)  
*smoother type*
- [SHORT smooth\\_order](#)  
*smoother order*
- [SHORT presmooth\\_iter](#)  
*number of presmoothers*
- [SHORT postsmooth\\_iter](#)  
*number of postsmoothers*
- [REAL relaxation](#)  
*relaxation parameter for Jacobi and SOR smoother*
- [SHORT polynomial\\_degree](#)  
*degree of the polynomial smoother*
- [SHORT coarse\\_solver](#)  
*coarse solver type*
- [SHORT coarse\\_scaling](#)  
*switch of scaling of the coarse grid correction*
- [SHORT aml\\_i\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [REAL \\* aml\\_i\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [SHORT nl\\_aml\\_i\\_krylov\\_type](#)  
*type of Krylov method used by Nonlinear AMLI cycle*
- [SHORT coarsening\\_type](#)  
*coarsening type*
- [SHORT aggregation\\_type](#)  
*aggregation type*
- [SHORT interpolation\\_type](#)  
*interpolation type*
- [REAL strong\\_threshold](#)  
*strong connection threshold for coarsening*
- [REAL max\\_row\\_sum](#)  
*maximal row sum parameter*
- [REAL truncation\\_threshold](#)  
*truncation threshold*
- [INT aggressive\\_level](#)  
*number of levels use aggressive coarsening*
- [INT aggressive\\_path](#)  
*number of paths use to determine strongly coupled C points*
- [INT pair\\_number](#)  
*number of pairwise matchings*
- [REAL strong\\_coupled](#)  
*strong coupled threshold for aggregate*
- [INT max\\_aggregation](#)

- max size of each aggregate*
- [REAL tentative\\_smooth](#)  
*relaxation parameter for smoothing the tentative prolongation*
- [SHORT smooth\\_filter](#)  
*switch for filtered matrix used for smoothing the tentative prolongation*
- [SHORT smooth\\_restriction](#)  
*smooth the restriction for SA methods or not*
- [SHORT ILU\\_levels](#)  
*number of levels use ILU smoother*
- [SHORT ILU\\_type](#)  
*ILU type for smoothing.*
- [INT ILU\\_lfil](#)  
*level of fill-in for ILUs and ILUK*
- [REAL ILU\\_droptol](#)  
*drop tolerance for ILUt*
- [REAL ILU\\_relax](#)  
*relaxation for ILUs*
- [REAL ILU\\_permtol](#)  
*permuted if  $\text{permtol} * |a(i,j)| > |a(i,i)|$*
- [INT SWZ\\_levels](#)  
*number of levels use Schwarz smoother*
- [INT SWZ\\_mmsize](#)  
*maximal block size*
- [INT SWZ\\_maxlvl](#)  
*maximal levels*
- [INT SWZ\\_type](#)  
*type of Schwarz method*
- [INT SWZ\\_blksolver](#)  
*type of Schwarz block solver*

### 8.3.1 Detailed Description

Parameters for AMG methods.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 434 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)



## 8.4 `block_dvector` Struct Reference

Block REAL vector structure.

```
#include <fasp_block.h>
```

### Data Fields

- `INT brow`  
*row number of blocks in A, m*
- `dvector ** blocks`  
*blocks of dvector, point to blocks[brow]*

### 8.4.1 Detailed Description

Block REAL vector structure.

Definition at line 110 of file `fasp_block.h`.

The documentation for this struct was generated from the following file:

- `fasp_block.h`

## 8.5 `block_ivec` Struct Reference

Block INT vector structure.

```
#include <fasp_block.h>
```

### Data Fields

- `INT brow`  
*row number of blocks in A, m*
- `ivec ** blocks`  
*blocks of dvector, point to blocks[brow]*

### 8.5.1 Detailed Description

Block INT vector structure.

#### Note

The starting index of A is 0.

Definition at line 126 of file `fasp_block.h`.

The documentation for this struct was generated from the following file:

- `fasp_block.h`

## 8.6 dBLCmat Struct Reference

Block REAL CSR matrix format.

```
#include <fasp_block.h>
```

### Data Fields

- [INT brow](#)  
*row number of blocks in A, m*
- [INT bcol](#)  
*column number of blocks A, n*
- [dCSRmat \\*\\* blocks](#)  
*blocks of [dCSRmat](#), point to blocks[brow][bcol]*

### 8.6.1 Detailed Description

Block REAL CSR matrix format.

Note

The starting index of A is 0.

Definition at line 74 of file fasp\_block.h.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.7 dBSRmat Struct Reference

Block sparse row storage matrix of REAL type.

```
#include <fasp_block.h>
```

### Data Fields

- [INT ROW](#)  
*number of rows of sub-blocks in matrix A, M*
- [INT COL](#)  
*number of cols of sub-blocks in matrix A, N*
- [INT NNZ](#)  
*number of nonzero sub-blocks in matrix A, NNZ*
- [INT nb](#)  
*dimension of each sub-block*
- [INT storage\\_manner](#)  
*storage manner for each sub-block*
- [REAL \\* val](#)
- [INT \\* IA](#)  
*integer array of row pointers, the size is ROW+1*
- [INT \\* JA](#)

### 8.7.1 Detailed Description

Block sparse row storage matrix of REAL type.

#### Note

This data structure is adapted from the Intel MKL library. Refer to: <http://software.intel.com/sites/products/documentation/hpc/mkl/lin/index.htm>

Some of the following entries are capitalized to stress that they are for blocks!

Definition at line 34 of file fasp\_block.h.

### 8.7.2 Field Documentation

#### 8.7.2.1 JA

`INT* JA`

Element  $i$  of the integer array columns is the number of the column in the block matrix that contains the  $i$ -th non-zero block. The size is NNZ.

Definition at line 64 of file fasp\_block.h.

#### 8.7.2.2 val

`REAL* val`

A real array that contains the elements of the non-zero blocks of a sparse matrix. The elements are stored block-by-block in row major order. A non-zero block is the block that contains at least one non-zero element. All elements of non-zero blocks are stored, even if some of them is equal to zero. Within each nonzero block elements are stored in row-major order and the size is  $(\text{NNZ} \times \text{nb} \times \text{nb})$ .

Definition at line 57 of file fasp\_block.h.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.8 dCOOmat Struct Reference

Sparse matrix of REAL type in COO (IJ) format.

```
#include <fasp.h>
```

### Data Fields

- [INT row](#)  
*row number of matrix A, m*
- [INT col](#)  
*column of matrix A, n*
- [INT nnz](#)  
*number of nonzero entries*
- [INT \\* rowind](#)  
*integer array of row indices, the size is nnz*
- [INT \\* colind](#)  
*integer array of column indices, the size is nnz*
- [REAL \\* val](#)  
*nonzero entries of A*

### 8.8.1 Detailed Description

Sparse matrix of REAL type in COO (IJ) format.

Coordinate Format (I,J,A)

#### Note

The starting index of A is 0.

Change I to rowind, J to colind. To avoid with complex.h confliction on I.

Definition at line 201 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.9 dCSRLmat Struct Reference

Sparse matrix of REAL type in CSRL format.

```
#include <fasp.h>
```

## Data Fields

- [INT row](#)  
*number of rows*
- [INT col](#)  
*number of cols*
- [INT nnz](#)  
*number of nonzero entries*
- [INT dif](#)  
*number of different values in i-th row, i=0:nrows-1*
- [INT \\* nz\\_diff](#)  
*nz\_diff[i]: the i-th different value in 'nzrow'*
- [INT \\* index](#)  
*row index of the matrix (length-grouped): rows with same nnz are together*
- [INT \\* start](#)  
*j in {start[i],...,start[i+1]-1} means nz\_diff[i] nnz in index[j]-row*
- [INT \\* ja](#)  
*column indices of all the nonzeros*
- [REAL \\* val](#)  
*values of all the nonzero entries*

### 8.9.1 Detailed Description

Sparse matrix of REAL type in CSRL format.

Definition at line 257 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.10 dCSRmat Struct Reference

Sparse matrix of REAL type in CSR format.

```
#include <fasp.h>
```

## Data Fields

- [INT row](#)  
*row number of matrix A, m*
- [INT col](#)  
*column of matrix A, n*
- [INT nnz](#)  
*number of nonzero entries*
- [INT \\* IA](#)  
*integer array of row pointers, the size is m+1*
- [INT \\* JA](#)  
*integer array of column indexes, the size is nnz*
- [REAL \\* val](#)  
*nonzero entries of A*

### 8.10.1 Detailed Description

Sparse matrix of REAL type in CSR format.

CSR Format (IA,JA,A) in REAL

#### Note

The starting index of A is 0.

Definition at line 140 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.11 ddenmat Struct Reference

Dense matrix of REAL type.

```
#include <fasp.h>
```

### Data Fields

- [INT row](#)  
*number of rows*
- [INT col](#)  
*number of columns*
- [REAL \\*\\* val](#)  
*actual matrix entries*

### 8.11.1 Detailed Description

Dense matrix of REAL type.

A dense REAL matrix

Definition at line 100 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.12 dSTRmat Struct Reference

Structure matrix of REAL type.

```
#include <fasp.h>
```

### Data Fields

- [INT nx](#)  
*number of grids in x direction*
- [INT ny](#)  
*number of grids in y direction*
- [INT nz](#)  
*number of grids in z direction*
- [INT nxy](#)  
*number of grids on x-y plane*
- [INT nc](#)  
*size of each block (number of components)*
- [INT ngrid](#)  
*number of grids*
- [REAL \\* diag](#)  
*diagonal entries (length is  $ngrid * (nc^2)$ )*
- [INT nband](#)  
*number of off-diag bands*
- [INT \\* offsets](#)  
*offsets of the off-diagonals (length is nband)*
- [REAL \\*\\* offdiag](#)  
*off-diagonal entries (dimension is  $nband * [(ngrid - |offsets|) * nc^2]$ )*

### 8.12.1 Detailed Description

Structure matrix of REAL type.

#### Note

Every  $nc^2$  entries of the array `diag` and `off-diag[i]` store one block: For 2D matrix, the recommended offsets is `[-1,1,-nx,nx]`; For 3D matrix, the recommended offsets is `[-1,1,-nx,nx,-nxy,nxy]`.

Definition at line 296 of file `fasp.h`.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.13 dvector Struct Reference

Vector with n entries of REAL type.

```
#include <fasp.h>
```

### Data Fields

- [INT row](#)  
*number of rows*
- [REAL \\* val](#)  
*actual vector entries*

### 8.13.1 Detailed Description

Vector with n entries of REAL type.

Definition at line 334 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.14 grid2d Struct Reference

Two dimensional grid data structure.

```
#include <fasp_grid.h>
```

### Data Fields

- [REAL\(\\* p\)\[2\]](#)
- [INT\(\\* e\)\[2\]](#)
- [INT\(\\* t\)\[3\]](#)
- [INT\(\\* s\)\[3\]](#)
- [INT \\* pdiri](#)
- [INT \\* ediri](#)
- [INT \\* pfather](#)
- [INT \\* efather](#)
- [INT \\* tfather](#)
- [INT vertices](#)
- [INT edges](#)
- [INT triangles](#)



### 8.14.1 Detailed Description

Two dimensional grid data structure.

#### Note

The `grid2d` structure is simply a list of triangles, edges and vertices. edge  $i$  has 2 vertices  $e[i]$ , triangle  $i$  has 3 edges  $s[i]$ , 3 vertices  $t[i]$  vertex  $i$  has two coordinates  $p[i]$

Definition at line 24 of file `fasp_grid.h`.

### 8.14.2 Field Documentation

#### 8.14.2.1 `e`

```
INT (* e) [2]
```

Vertices of edges

Definition at line 27 of file `fasp_grid.h`.

#### 8.14.2.2 `edges`

```
INT edges
```

Number of edges

Definition at line 38 of file `fasp_grid.h`.

#### 8.14.2.3 `ediri`

```
INT* ediri
```

Boundary flags (0 <=> interior edge)

Definition at line 31 of file `fasp_grid.h`.

#### 8.14.2.4 efather

`INT* efather`

Father edge or triangle

Definition at line 34 of file fasp\_grid.h.

#### 8.14.2.5 p

`REAL(* p) [2]`

Coordinates of vertices

Definition at line 26 of file fasp\_grid.h.

#### 8.14.2.6 pdiri

`INT* pdiri`

Boundary flags (0 <=> interior point)

Definition at line 30 of file fasp\_grid.h.

#### 8.14.2.7 pfather

`INT* pfather`

Father point or edge

Definition at line 33 of file fasp\_grid.h.

#### 8.14.2.8 s

`INT(* s) [3]`

Edges of triangles

Definition at line 29 of file fasp\_grid.h.

#### 8.14.2.9 t

```
INT (* t) [3]
```

Vertices of triangles

Definition at line 28 of file fasp\_grid.h.

#### 8.14.2.10 tfather

```
INT* tfather
```

Father triangle

Definition at line 35 of file fasp\_grid.h.

#### 8.14.2.11 triangles

```
INT triangles
```

Number of triangles

Definition at line 39 of file fasp\_grid.h.

#### 8.14.2.12 vertices

```
INT vertices
```

Number of grid points

Definition at line 37 of file fasp\_grid.h.

The documentation for this struct was generated from the following file:

- [fasp\\_grid.h](#)

## 8.15 iBLCmat Struct Reference

Block INT CSR matrix format.

```
#include <fasp_block.h>
```

## Data Fields

- [INT brow](#)  
*row number of blocks in A, m*
- [INT bcol](#)  
*column number of blocks A, n*
- [iCSRmat \\*\\* blocks](#)  
*blocks of [iCSRmat](#), point to blocks[brow][bcol]*

### 8.15.1 Detailed Description

Block INT CSR matrix format.

#### Note

The starting index of A is 0.

Definition at line 93 of file fasp\_block.h.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.16 iCOOmat Struct Reference

Sparse matrix of INT type in COO (IJ) format.

```
#include <fasp.h>
```

## Data Fields

- [INT row](#)  
*row number of matrix A, m*
- [INT col](#)  
*column of matrix A, n*
- [INT nnz](#)  
*number of nonzero entries*
- [INT \\* I](#)  
*integer array of row indices, the size is nnz*
- [INT \\* J](#)  
*integer array of column indices, the size is nnz*
- [INT \\* val](#)  
*nonzero entries of A*

### 8.16.1 Detailed Description

Sparse matrix of INT type in COO (IJ) format.

Coordinate Format (I,J,A)

#### Note

The starting index of A is 0.

Definition at line 231 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.17 iCSRmat Struct Reference

Sparse matrix of INT type in CSR format.

```
#include <fasp.h>
```

### Data Fields

- [INT row](#)  
*row number of matrix A, m*
- [INT col](#)  
*column of matrix A, n*
- [INT nnz](#)  
*number of nonzero entries*
- [INT \\* IA](#)  
*integer array of row pointers, the size is m+1*
- [INT \\* JA](#)  
*integer array of column indexes, the size is nnz*
- [INT \\* val](#)  
*nonzero entries of A*

### 8.17.1 Detailed Description

Sparse matrix of INT type in CSR format.

CSR Format (IA,JA,A) in integer

#### Note

The starting index of A is 0.

Definition at line 170 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.18 idenmat Struct Reference

Dense matrix of INT type.

```
#include <fasp.h>
```

### Data Fields

- [INT row](#)  
*number of rows*
- [INT col](#)  
*number of columns*
- [INT \\*\\* val](#)  
*actual matrix entries*

### 8.18.1 Detailed Description

Dense matrix of INT type.

A dense INT matrix

Definition at line 119 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.19 ILU\_data Struct Reference

Data for ILU setup.

```
#include <fasp.h>
```

## Data Fields

- [dCSRmat](#) \* [A](#)  
*pointer to the original coefficient matrix*
- [INT](#) [type](#)  
*type of ILUdata*
- [INT](#) [row](#)  
*row number of matrix LU, m*
- [INT](#) [col](#)  
*column of matrix LU, n*
- [INT](#) [nzlu](#)  
*number of nonzero entries*
- [INT](#) \* [ijlu](#)  
*integer array of row pointers and column indexes, the size is nzlu*
- [REAL](#) \* [luval](#)  
*nonzero entries of LU*
- [INT](#) [nb](#)  
*block size for BSR type only*
- [INT](#) [nwork](#)  
*work space size*
- [REAL](#) \* [work](#)  
*work space*
- [INT](#) \* [iperm](#)  
*permutation arrays for ILUtp*
- [INT](#) [ncolors](#)  
*number of colors for multi-threading*
- [INT](#) \* [ic](#)  
*indices for different colors*
- [INT](#) \* [icmap](#)  
*mapping from vertex to color*
- [INT](#) \* [uptr](#)  
*temporary work space*
- [INT](#) [nlevL](#)  
*number of colors for lower triangle*
- [INT](#) [nlevU](#)  
*number of colors for upper triangle*
- [INT](#) \* [ilevL](#)  
*number of vertices in each color for lower triangle*
- [INT](#) \* [ilevU](#)  
*number of vertices in each color for upper triangle*
- [INT](#) \* [jlevL](#)  
*mapping from row to color for lower triangle*
- [INT](#) \* [jlevU](#)  
*mapping from row to color for upper triangle*

### 8.19.1 Detailed Description

Data for ILU setup.

Definition at line 624 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.20 ILU\_param Struct Reference

Parameters for ILU.

```
#include <fasp.h>
```

### Data Fields

- [SHORT print\\_level](#)  
*print level*
- [SHORT ILU\\_type](#)  
*ILU type for decomposition.*
- [INT ILU\\_lfil](#)  
*level of fill-in for ILUk*
- [REAL ILU\\_droptol](#)  
*drop tolerance for ILUt*
- [REAL ILU\\_relax](#)  
*add the sum of dropped elements to diagonal element in proportion relax*
- [REAL ILU\\_permtol](#)  
*permuted if  $\text{permtol} * |a(i,j)| > |a(i,i)|$*

### 8.20.1 Detailed Description

Parameters for ILU.

Definition at line 383 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)



## 8.21 input\_param Struct Reference

Input parameters.

```
#include <fasp.h>
```

### Data Fields

- [SHORT print\\_level](#)
- [SHORT output\\_type](#)
- [char inifile \[STRLEN\]](#)
- [char workdir \[STRLEN\]](#)
- [INT problem\\_num](#)
- [SHORT solver\\_type](#)
- [SHORT decoup\\_type](#)
- [SHORT precondition\\_type](#)
- [SHORT stop\\_type](#)
- [REAL itsolver\\_tol](#)
- [INT itsolver\\_maxit](#)
- [INT restart](#)
- [SHORT ILU\\_type](#)
- [INT ILU\\_lfil](#)
- [REAL ILU\\_droptol](#)
- [REAL ILU\\_relax](#)
- [REAL ILU\\_permtol](#)
- [INT SWZ\\_mmsize](#)
- [INT SWZ\\_maxlvl](#)
- [INT SWZ\\_type](#)
- [INT SWZ\\_blksolver](#)
- [SHORT AMG\\_type](#)
- [SHORT AMG\\_levels](#)
- [SHORT AMG\\_cycle\\_type](#)
- [SHORT AMG\\_smoother](#)
- [SHORT AMG\\_smooth\\_order](#)
- [REAL AMG\\_relaxation](#)
- [SHORT AMG\\_polynomial\\_degree](#)
- [SHORT AMG\\_presmooth\\_iter](#)
- [SHORT AMG\\_postsmooth\\_iter](#)
- [REAL AMG\\_tol](#)
- [INT AMG\\_coarse\\_dof](#)
- [INT AMG\\_maxit](#)
- [SHORT AMG\\_ILU\\_levels](#)
- [SHORT AMG\\_coarse\\_solver](#)
- [SHORT AMG\\_coarse\\_scaling](#)
- [SHORT AMG\\_amli\\_degree](#)
- [SHORT AMG\\_nl\\_amli\\_krylov\\_type](#)
- [INT AMG\\_SWZ\\_levels](#)
- [SHORT AMG\\_coarsening\\_type](#)
- [SHORT AMG\\_aggregation\\_type](#)

- [SHORT AMG\\_interpolation\\_type](#)
- [REAL AMG\\_strong\\_threshold](#)
- [REAL AMG\\_truncation\\_threshold](#)
- [REAL AMG\\_max\\_row\\_sum](#)
- [INT AMG\\_aggressive\\_level](#)
- [INT AMG\\_aggressive\\_path](#)
- [INT AMG\\_pair\\_number](#)
- [REAL AMG\\_quality\\_bound](#)
- [REAL AMG\\_strong\\_coupled](#)
- [INT AMG\\_max\\_aggregation](#)
- [REAL AMG\\_tentative\\_smooth](#)
- [SHORT AMG\\_smooth\\_filter](#)
- [SHORT AMG\\_smooth\\_restriction](#)

### 8.21.1 Detailed Description

Input parameters.

Input parameters, reading from disk file

Definition at line 1093 of file fasp.h.

### 8.21.2 Field Documentation

#### 8.21.2.1 AMG\_aggregation\_type

[SHORT](#) AMG\_aggregation\_type

aggregation type

Definition at line 1148 of file fasp.h.

#### 8.21.2.2 AMG\_aggressive\_level

[INT](#) AMG\_aggressive\_level

number of levels use aggressive coarsening

Definition at line 1153 of file fasp.h.

### 8.21.2.3 AMG\_aggressive\_path

`INT AMG_aggressive_path`

number of paths to determine strongly coupled C-set

Definition at line 1154 of file fasp.h.

### 8.21.2.4 AMG\_amli\_degree

`SHORT AMG_amli_degree`

degree of the polynomial used by AMLI cycle

Definition at line 1142 of file fasp.h.

### 8.21.2.5 AMG\_coarse\_dof

`INT AMG_coarse_dof`

max number of coarsest level DOF

Definition at line 1137 of file fasp.h.

### 8.21.2.6 AMG\_coarse\_scaling

`SHORT AMG_coarse_scaling`

switch of scaling of the coarse grid correction

Definition at line 1141 of file fasp.h.

### 8.21.2.7 AMG\_coarse\_solver

`SHORT AMG_coarse_solver`

coarse solver type

Definition at line 1140 of file fasp.h.

#### 8.21.2.8 AMG\_coarsening\_type

`SHORT` AMG\_coarsening\_type

coarsening type

Definition at line 1147 of file fasp.h.

#### 8.21.2.9 AMG\_cycle\_type

`SHORT` AMG\_cycle\_type

type of cycle

Definition at line 1129 of file fasp.h.

#### 8.21.2.10 AMG\_ILU\_levels

`SHORT` AMG\_ILU\_levels

how many levels use ILU smoother

Definition at line 1139 of file fasp.h.

#### 8.21.2.11 AMG\_interpolation\_type

`SHORT` AMG\_interpolation\_type

interpolation type

Definition at line 1149 of file fasp.h.

#### 8.21.2.12 AMG\_levels

`SHORT` AMG\_levels

maximal number of levels

Definition at line 1128 of file fasp.h.

### 8.21.2.13 AMG\_max\_aggregation

`INT AMG_max_aggregation`

max size of each aggregate

Definition at line 1160 of file fasp.h.

### 8.21.2.14 AMG\_max\_row\_sum

`REAL AMG_max_row_sum`

maximal row sum

Definition at line 1152 of file fasp.h.

### 8.21.2.15 AMG\_maxit

`INT AMG_maxit`

number of iterations for AMG used as preconditioner

Definition at line 1138 of file fasp.h.

### 8.21.2.16 AMG\_nl\_amli\_krylov\_type

`SHORT AMG_nl_amli_krylov_type`

type of Krylov method used by nonlinear AMLI cycle

Definition at line 1143 of file fasp.h.

### 8.21.2.17 AMG\_pair\_number

`INT AMG_pair_number`

number of pairs in matching algorithm

Definition at line 1155 of file fasp.h.

**8.21.2.18 AMG\_polynomial\_degree**

`SHORT AMG_polynomial_degree`

degree of the polynomial smoother

Definition at line 1133 of file fasp.h.

**8.21.2.19 AMG\_postsmooth\_iter**

`SHORT AMG_postsmooth_iter`

number of postsmoothing

Definition at line 1135 of file fasp.h.

**8.21.2.20 AMG\_presmooth\_iter**

`SHORT AMG_presmooth_iter`

number of presmoothing

Definition at line 1134 of file fasp.h.

**8.21.2.21 AMG\_quality\_bound**

`REAL AMG_quality_bound`

threshold for pair wise aggregation

Definition at line 1156 of file fasp.h.

**8.21.2.22 AMG\_relaxation**

`REAL AMG_relaxation`

over-relaxation parameter for SOR

Definition at line 1132 of file fasp.h.

### 8.21.2.23 AMG\_smooth\_filter

`SHORT` AMG\_smooth\_filter

use filter for smoothing the tentative prolongation or not

Definition at line 1162 of file fasp.h.

### 8.21.2.24 AMG\_smooth\_order

`SHORT` AMG\_smooth\_order

order for smoothers

Definition at line 1131 of file fasp.h.

### 8.21.2.25 AMG\_smooth\_restriction

`SHORT` AMG\_smooth\_restriction

smoothing the restriction or not

Definition at line 1163 of file fasp.h.

### 8.21.2.26 AMG\_smoother

`SHORT` AMG\_smoother

type of smoother

Definition at line 1130 of file fasp.h.

### 8.21.2.27 AMG\_strong\_coupled

`REAL` AMG\_strong\_coupled

strong coupled threshold for aggregate

Definition at line 1159 of file fasp.h.

**8.21.2.28 AMG\_strong\_threshold**

`REAL AMG_strong_threshold`

strong threshold for coarsening

Definition at line 1150 of file fasp.h.

**8.21.2.29 AMG\_SWZ\_levels**

`INT AMG_SWZ_levels`

number of levels use Schwarz smoother

Definition at line 1144 of file fasp.h.

**8.21.2.30 AMG\_tentative\_smooth**

`REAL AMG_tentative_smooth`

relaxation factor for smoothing the tentative prolongation

Definition at line 1161 of file fasp.h.

**8.21.2.31 AMG\_tol**

`REAL AMG_tol`

tolerance for AMG if used as preconditioner

Definition at line 1136 of file fasp.h.

**8.21.2.32 AMG\_truncation\_threshold**

`REAL AMG_truncation_threshold`

truncation factor for interpolation

Definition at line 1151 of file fasp.h.



### 8.21.2.33 AMG\_type

`SHORT` AMG\_type

Type of AMG

Definition at line 1127 of file fasp.h.

### 8.21.2.34 decoup\_type

`SHORT` decoup\_type

type of decoupling method for PDE systems

Definition at line 1106 of file fasp.h.

### 8.21.2.35 ILU\_droptol

`REAL` ILU\_droptol

drop tolerance

Definition at line 1116 of file fasp.h.

### 8.21.2.36 ILU\_lfil

`INT` ILU\_lfil

level of fill-in

Definition at line 1115 of file fasp.h.

### 8.21.2.37 ILU\_permtol

`REAL` ILU\_permtol

permutation tolerance

Definition at line 1118 of file fasp.h.

#### 8.21.2.38 ILU\_relax

`REAL ILU_relax`

scaling factor: add the sum of dropped entries to diagonal

Definition at line 1117 of file fasp.h.

#### 8.21.2.39 ILU\_type

`SHORT ILU_type`

ILU type for decomposition

Definition at line 1114 of file fasp.h.

#### 8.21.2.40 inifile

`char inifile[STRLEN]`

ini file name

Definition at line 1100 of file fasp.h.

#### 8.21.2.41 itsolver\_maxit

`INT itsolver_maxit`

maximal number of iterations for iterative solvers

Definition at line 1110 of file fasp.h.

#### 8.21.2.42 itsolver\_tol

`REAL itsolver_tol`

tolerance for iterative linear solver

Definition at line 1109 of file fasp.h.

#### 8.21.2.43 output\_type

`SHORT output_type`

type of output stream

Definition at line 1097 of file fasp.h.

#### 8.21.2.44 precondition\_type

`SHORT precondition_type`

type of preconditioner for iterative solvers

Definition at line 1107 of file fasp.h.

#### 8.21.2.45 print\_level

`SHORT print_level`

print level

Definition at line 1096 of file fasp.h.

#### 8.21.2.46 problem\_num

`INT problem_num`

problem number to solve

Definition at line 1102 of file fasp.h.

#### 8.21.2.47 restart

`INT restart`

restart number used in GMRES

Definition at line 1111 of file fasp.h.

**8.21.2.48 solver\_type**

`SHORT solver_type`

type of iterative solvers

Definition at line 1105 of file fasp.h.

**8.21.2.49 stop\_type**

`SHORT stop_type`

type of stopping criteria for iterative solvers

Definition at line 1108 of file fasp.h.

**8.21.2.50 SWZ\_blksolver**

`INT SWZ_blksolver`

type of Schwarz block solver

Definition at line 1124 of file fasp.h.

**8.21.2.51 SWZ\_maxlvl**

`INT SWZ_maxlvl`

maximal levels

Definition at line 1122 of file fasp.h.

**8.21.2.52 SWZ\_mmsize**

`INT SWZ_mmsize`

maximal block size

Definition at line 1121 of file fasp.h.

### 8.21.2.53 SWZ\_type

`INT` `SWZ_type`

type of Schwarz method

Definition at line 1123 of file fasp.h.

### 8.21.2.54 workdir

`char` `workdir`[`STRLEN`]

working directory for data files

Definition at line 1101 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.22 ITS\_param Struct Reference

Parameters for iterative solvers.

```
#include <fasp.h>
```

### Data Fields

- [SHORT](#) `print_level`
- [SHORT](#) `itsolver_type`
- [SHORT](#) `decoup_type`
- [SHORT](#) `precond_type`
- [SHORT](#) `stop_type`
- [INT](#) `restart`
- [INT](#) `maxit`
- [REAL](#) `tol`

### 8.22.1 Detailed Description

Parameters for iterative solvers.

Definition at line 366 of file fasp.h.

## 8.22.2 Field Documentation

### 8.22.2.1 decoup\_type

`SHORT decoup_type`

decoupling type

Definition at line 370 of file fasp.h.

### 8.22.2.2 itsolver\_type

`SHORT itsolver_type`

solver type: see [fasp\\_const.h](#)

Definition at line 369 of file fasp.h.

### 8.22.2.3 maxit

`INT maxit`

max number of iterations

Definition at line 374 of file fasp.h.

### 8.22.2.4 precondition\_type

`SHORT precondition_type`

preconditioner type

Definition at line 371 of file fasp.h.

#### 8.22.2.5 print\_level

`SHORT print_level`

print level: 0–10

Definition at line 368 of file fasp.h.

#### 8.22.2.6 restart

`INT restart`

number of steps for restarting: for GMRES etc

Definition at line 373 of file fasp.h.

#### 8.22.2.7 stop\_type

`SHORT stop_type`

stopping type

Definition at line 372 of file fasp.h.

#### 8.22.2.8 tol

`REAL tol`

convergence tolerance

Definition at line 375 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.23 ivector Struct Reference

Vector with n entries of INT type.

```
#include <fasp.h>
```

## Data Fields

- [INT row](#)  
*number of rows*
- [INT \\* val](#)  
*actual vector entries*

### 8.23.1 Detailed Description

Vector with n entries of INT type.

Definition at line 348 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.24 Mumps\_data Struct Reference

Data for MUMPS interface.

```
#include <fasp.h>
```

## Data Fields

- [INT job](#)  
*work for MUMPS*

### 8.24.1 Detailed Description

Data for MUMPS interface.

Added on 10/10/2014

Definition at line 580 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)



## 8.25 mxv\_matfree Struct Reference

Matrix-vector multiplication, replace the actual matrix.

```
#include <fasp.h>
```

### Data Fields

- void \* [data](#)  
*data for MxV, can be a Matrix or something else*
- void(\* [fct](#))(const void \*, const [REAL](#) \*, [REAL](#) \*)  
*action for MxV, void function pointer*

### 8.25.1 Detailed Description

Matrix-vector multiplication, replace the actual matrix.

Definition at line 1077 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.26 Pardiso\_data Struct Reference

Data for Intel MKL PARDISO interface.

```
#include <fasp.h>
```

### Data Fields

- void \* [pt](#) [64]  
*Internal solver memory pointer.*

### 8.26.1 Detailed Description

Data for Intel MKL PARDISO interface.

Added on 11/28/2015

Definition at line 598 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.27 precondition Struct Reference

Preconditioner data and action.

```
#include <fasp.h>
```

### Data Fields

- void \* [data](#)  
*data for preconditioner, void pointer*
- void(\* [fct](#))([REAL](#) \*, [REAL](#) \*, void \*)  
*action for preconditioner, void function pointer*

### 8.27.1 Detailed Description

Preconditioner data and action.

#### Note

This is the preconditioner structure for preconditioned iterative methods.

Definition at line 1063 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.28 precondition\_data Struct Reference

Data for preconditioners.

```
#include <fasp.h>
```

## Data Fields

- [SHORT AMG\\_type](#)  
*type of AMG method*
- [SHORT print\\_level](#)  
*print level in AMG preconditioner*
- [INT maxit](#)  
*max number of iterations of AMG preconditioner*
- [SHORT max\\_levels](#)  
*max number of AMG levels*
- [REAL tol](#)  
*tolerance for AMG preconditioner*
- [SHORT cycle\\_type](#)  
*AMG cycle type.*
- [SHORT smoother](#)  
*AMG smoother type.*
- [SHORT smooth\\_order](#)  
*AMG smoother ordering.*
- [SHORT presmooth\\_iter](#)  
*number of presmoothing*
- [SHORT postsmooth\\_iter](#)  
*number of postsmoothing*
- [REAL relaxation](#)  
*relaxation parameter for SOR smoother*
- [SHORT polynomial\\_degree](#)  
*degree of the polynomial smoother*
- [SHORT coarsening\\_type](#)  
*switch of scaling of the coarse grid correction*
- [SHORT coarse\\_solver](#)  
*coarse solver type for AMG*
- [SHORT coarse\\_scaling](#)  
*switch of scaling of the coarse grid correction*
- [SHORT amli\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [SHORT nl\\_amli\\_krylov\\_type](#)  
*type of Krylov method used by Nonlinear AMLI cycle*
- [REAL tentative\\_smooth](#)  
*smooth factor for smoothing the tentative prolongation*
- [REAL \\* amli\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [AMG\\_data \\* mgl\\_data](#)  
*AMG preconditioner data.*
- [ILU\\_data \\* LU](#)  
*ILU preconditioner data (needed for CPR type preconditioner)*
- [dCSRmat \\* A](#)  
*Matrix data.*
- [dCSRmat \\* A\\_nk](#)

- *Matrix data for near kernel.*  
• [dCSRmat](#) \* [P\\_nk](#)  
• *Prolongation for near kernel.*  
• [dCSRmat](#) \* [R\\_nk](#)  
• *Restriction for near kernel.*  
• [dvector](#) [r](#)  
• *temporary dvector used to store and restore the residual*  
• [REAL](#) \* [w](#)  
• *temporary work space for other usage*

### 8.28.1 Detailed Description

Data for preconditioners.

Definition at line 862 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.29 precondition\_data\_blc Struct Reference

Data for block preconditioners in [dBLCmat](#) format.

```
#include <fasp_block.h>
```

### Data Fields

- [dBLCmat](#) \* [Ablc](#)
- [dCSRmat](#) \* [A\\_diag](#)
- [dvector](#) [r](#)
- void \*\* [LU\\_diag](#)
- [AMG\\_data](#) \*\* [mgl](#)
- [AMG\\_param](#) \* [amgparam](#)

### 8.29.1 Detailed Description

Data for block preconditioners in [dBLCmat](#) format.

This is needed for the block preconditioner.

Definition at line 349 of file fasp\_block.h.

## 8.29.2 Field Documentation

### 8.29.2.1 A\_diag

`dCSRmat* A_diag`

data for each diagonal block

Definition at line 356 of file fasp\_block.h.

### 8.29.2.2 Ablc

`dBLMat* Ablc`

problem data, the blocks

Definition at line 354 of file fasp\_block.h.

### 8.29.2.3 amgparam

`AMG_param* amgparam`

parameters for AMG

Definition at line 370 of file fasp\_block.h.

### 8.29.2.4 LU\_diag

`void** LU_diag`

LU decomposition for the diagonal blocks (for UMFPack)

Definition at line 365 of file fasp\_block.h.

### 8.29.2.5 mgl

`AMG_data** mgl`

AMG data for the diagonal blocks

Definition at line 368 of file `fasp_block.h`.

### 8.29.2.6 r

`dvector r`

temp work space

Definition at line 358 of file `fasp_block.h`.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.30 precond\_data\_bsr Struct Reference

Data for preconditioners in [dBSRmat](#) format.

```
#include <fasp_block.h>
```

### Data Fields

- [SHORT AMG\\_type](#)  
*type of AMG method*
- [SHORT print\\_level](#)  
*print level in AMG preconditioner*
- [INT maxit](#)  
*max number of iterations of AMG preconditioner*
- [INT max\\_levels](#)  
*max number of AMG levels*
- [REAL tol](#)  
*tolerance for AMG preconditioner*
- [SHORT cycle\\_type](#)  
*AMG cycle type.*
- [SHORT smoother](#)  
*AMG smoother type.*
- [SHORT smooth\\_order](#)

- AMG smoother ordering.
- [SHORT presmooth\\_iter](#)
  - number of presmoothing
- [SHORT postsmooth\\_iter](#)
  - number of postsmoothing
- [SHORT coarsening\\_type](#)
  - coarsening type
- [REAL relaxation](#)
  - relaxation parameter for SOR smoother
- [SHORT coarse\\_solver](#)
  - coarse solver type for AMG
- [SHORT coarse\\_scaling](#)
  - switch of scaling of the coarse grid correction
- [SHORT amli\\_degree](#)
  - degree of the polynomial used by AMLI cycle
- [REAL \\* amli\\_coef](#)
  - coefficients of the polynomial used by AMLI cycle
- [REAL tentative\\_smooth](#)
  - smooth factor for smoothing the tentative prolongation
- [SHORT nl\\_amli\\_krylov\\_type](#)
  - type of krylov method used by Nonlinear AMLI cycle
- [AMG\\_data\\_bsr \\* mgl\\_data](#)
  - AMG preconditioner data.
- [AMG\\_data \\* pres\\_mgl\\_data](#)
  - AMG preconditioner data for pressure block.
- [ILU\\_data \\* LU](#)
  - ILU preconditioner data (needed for CPR type preconditioner)
- [dBSRmat \\* A](#)
  - Matrix data.
- [dCSRmat \\* A\\_nk](#)
  - Matrix data for near kernal.
- [dCSRmat \\* P\\_nk](#)
  - Prolongation for near kernal.
- [dCSRmat \\* R\\_nk](#)
  - Resriction for near kernal.
- [dvector r](#)
  - temporary dvector used to store and restore the residual
- [REAL \\* w](#)
  - temporary work space for other usage

### 8.30.1 Detailed Description

Data for preconditioners in [dBSRmat](#) format.

**Note**

This structure is needed for the AMG solver/preconditioner in BSR format

Definition at line 257 of file fasp\_block.h.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.31 precond\_data\_str Struct Reference

Data for preconditioners in [dSTRmat](#) format.

```
#include <fasp.h>
```

### Data Fields

- [SHORT AMG\\_type](#)  
*type of AMG method*
- [SHORT print\\_level](#)  
*print level in AMG preconditioner*
- [INT maxit](#)  
*max number of iterations of AMG preconditioner*
- [SHORT max\\_levels](#)  
*max number of AMG levels*
- [REAL tol](#)  
*tolerance for AMG preconditioner*
- [SHORT cycle\\_type](#)  
*AMG cycle type.*
- [SHORT smoother](#)  
*AMG smoother type.*
- [SHORT presmooth\\_iter](#)  
*number of presmoothing*
- [SHORT postsmooth\\_iter](#)  
*number of postsmoothing*
- [SHORT coarsening\\_type](#)  
*coarsening type*
- [REAL relaxation](#)  
*relaxation parameter for SOR smoother*
- [SHORT coarse\\_scaling](#)  
*switch of scaling of the coarse grid correction*
- [AMG\\_data \\* mgl\\_data](#)  
*AMG preconditioner data.*
- [ILU\\_data \\* LU](#)



- ILU preconditioner data (needed for CPR type preconditioner)*
- [SHORT](#) scaled
  - whether the matrix are scaled or not*
- [dCSRmat](#) \* [A](#)
  - the original CSR matrix*
- [dSTRmat](#) \* [A\\_str](#)
  - store the whole reservoir block in STR format*
- [dSTRmat](#) \* [SS\\_str](#)
  - store Saturation block in STR format*
- [dvector](#) \* [diaginv](#)
  - the inverse of the diagonals for GS/block GS smoother (whole reservoir matrix)*
- [ivector](#) \* [pivot](#)
  - the pivot for the GS/block GS smoother (whole reservoir matrix)*
- [dvector](#) \* [diaginvS](#)
  - the inverse of the diagonals for GS/block GS smoother (saturation block)*
- [ivector](#) \* [pivotS](#)
  - the pivot for the GS/block GS smoother (saturation block)*
- [ivector](#) \* [order](#)
  - order for smoothing*
- [ivector](#) \* [neigh](#)
  - array to store neighbor information*
- [dvector](#) [r](#)
  - temporary dvector used to store and restore the residual*
- [REAL](#) \* [w](#)
  - temporary work space for other usage*

### 8.31.1 Detailed Description

Data for preconditioners in [dSTRmat](#) format.

Definition at line 955 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.32 precondition\_data\_sweeping Struct Reference

Data for sweeping preconditioner.

```
#include <fasp_block.h>
```

## Data Fields

- [INT NumLayers](#)
- [dBLMat \\* A](#)
- [dBLMat \\* Ai](#)
- [dCSRmat \\* local\\_A](#)
- [void \\*\\* local\\_LU](#)
- [ivector \\* local\\_index](#)
- [dvector r](#)
- [REAL \\* w](#)

### 8.32.1 Detailed Description

Data for sweeping preconditioner.

#### Author

Xiaozhe Hu

#### Date

05/01/2014

#### Note

This is needed for the sweeping preconditioner.

Definition at line 384 of file fasp\_block.h.

### 8.32.2 Field Documentation

#### 8.32.2.1 A

[dBLMat](#)\* A

problem data, the sparse matrix

Definition at line 388 of file fasp\_block.h.

### 8.32.2.2 Ai

`dBLMat*` Ai

preconditioner data, the sparse matrix

Definition at line 389 of file fasp\_block.h.

### 8.32.2.3 local\_A

`dCSRmat*` local\_A

local stiffness matrix for each layer

Definition at line 391 of file fasp\_block.h.

### 8.32.2.4 local\_index

`ivector*` local\_index

local index for each layer

Definition at line 394 of file fasp\_block.h.

### 8.32.2.5 local\_LU

`void**` local\_LU

local LU decomposition (for UMFPack)

Definition at line 392 of file fasp\_block.h.

### 8.32.2.6 NumLayers

`INT` NumLayers

number of layers

Definition at line 386 of file fasp\_block.h.

### 8.32.2.7 `r`

`dvector r`

temporary dvector used to store and restore the residual

Definition at line 397 of file `fasp_block.h`.

### 8.32.2.8 `w`

`REAL* w`

temporary work space for other usage

Definition at line 398 of file `fasp_block.h`.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.33 `precond_diag_bsr` Struct Reference

Data for diagonal preconditioners in `dBSRmat` format.

```
#include <fasp_block.h>
```

### Data Fields

- `INT nb`  
*dimension of each sub-block*
- `dvector diag`  
*diagonal elements*

#### 8.33.1 Detailed Description

Data for diagonal preconditioners in `dBSRmat` format.

##### Note

This is needed for the diagonal preconditioner.

Definition at line 241 of file `fasp_block.h`.

The documentation for this struct was generated from the following file:

- [fasp\\_block.h](#)

## 8.34 preconditioning\_struct Struct Reference

Data for diagonal preconditioners in [dSTRmat](#) format.

```
#include <fasp.h>
```

### Data Fields

- [INT nc](#)  
*number of components*
- [dvector diag](#)  
*diagonal elements*

#### 8.34.1 Detailed Description

Data for diagonal preconditioners in [dSTRmat](#) format.

##### Note

This is needed for the diagonal preconditioner.

Definition at line 1047 of file [fasp.h](#).

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.35 SWZ\_data Struct Reference

Data for Schwarz methods.

```
#include <fasp.h>
```

## Data Fields

- [dCSRmat A](#)  
*pointer to the original coefficient matrix*
- [INT nblk](#)  
*number of blocks*
- [INT \\* iblock](#)  
*row index of blocks*
- [INT \\* jblock](#)  
*column index of blocks*
- [REAL \\* rhsloc](#)  
*temp work space ???*
- [dvector rhsloc1](#)  
*local right hand side*
- [dvector xloc1](#)  
*local solution*
- [REAL \\* au](#)  
*LU decomposition: the U block.*
- [REAL \\* al](#)  
*LU decomposition: the L block.*
- [INT SWZ\\_type](#)  
*Schwarz method type.*
- [INT blk\\_solver](#)  
*Schwarz block solver.*
- [INT memt](#)  
*working space size*
- [INT \\* mask](#)  
*mask*
- [INT maxbs](#)  
*maximal block size*
- [INT \\* maxa](#)  
*maxa*
- [dCSRmat \\* blk\\_data](#)  
*matrix for each partition*
- [Mumps\\_data \\* mumps](#)  
*param for MUMPS*
- [SWZ\\_param \\* swzparam](#)  
*param for Schwarz*

### 8.35.1 Detailed Description

Data for Schwarz methods.

This is needed for the Schwarz solver/preconditioner/smoothers.

Definition at line 699 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)

## 8.36 SWZ\_param Struct Reference

Parameters for Schwarz method.

```
#include <fasp.h>
```

### Data Fields

- [SHORT print\\_level](#)  
*print leve*
- [SHORT SWZ\\_type](#)  
*type for Schwarz method*
- [INT SWZ\\_maxlvl](#)  
*maximal level for constructing the blocks*
- [INT SWZ\\_mmsize](#)  
*maximal size of blocks*
- [INT SWZ\\_blksolver](#)  
*type of Schwarz block solver*

### 8.36.1 Detailed Description

Parameters for Schwarz method.

Definition at line 409 of file fasp.h.

The documentation for this struct was generated from the following file:

- [fasp.h](#)





## Chapter 9

# File Documentation

### 9.1 AuxArray.c File Reference

Simple array operations – init, set, copy, etc.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

#### Functions

- void [fasp\\_darray\\_set](#) (const [INT](#) n, [REAL](#) \*x, const [REAL](#) val)  
*Set initial value for an array to be x=val.*
- void [fasp\\_iarray\\_set](#) (const [INT](#) n, [INT](#) \*x, const [INT](#) val)  
*Set initial value for an array to be x=val.*
- void [fasp\\_darray\\_cp](#) (const [INT](#) n, const [REAL](#) \*x, [REAL](#) \*y)  
*Copy an array to the other y=x.*
- void [fasp\\_iarray\\_cp](#) (const [INT](#) n, const [INT](#) \*x, [INT](#) \*y)  
*Copy an array to the other y=x.*

#### 9.1.1 Detailed Description

Simple array operations – init, set, copy, etc.

##### Note

This file contains Level-0 (Aux) functions. It requires: [AuxThreads.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.1.2 Function Documentation

### 9.1.2.1 fasp\_darray\_cp()

```
void fasp_darray_cp (
    const INT n,
    const REAL * x,
    REAL * y )
```

Copy an array to the other y=x.

#### Parameters

<i>n</i>	Number of variables
<i>x</i>	Pointer to the original vector
<i>y</i>	Pointer to the destination vector

#### Author

Chensong Zhang

#### Date

2010/04/03

Definition at line 164 of file AuxArray.c.

### 9.1.2.2 fasp\_darray\_set()

```
void fasp_darray_set (
    const INT n,
    REAL * x,
    const REAL val )
```

Set initial value for an array to be x=val.

#### Parameters

<i>n</i>	Number of variables
<i>x</i>	Pointer to the vector
<i>val</i>	Initial value for the REAL array

#### Author

Chensong Zhang

#### Date

04/03/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 41 of file AuxArray.c.

### 9.1.2.3 fasp\_iarray\_cp()

```
void fasp_iarray_cp (
    const INT n,
    const INT * x,
    INT * y )
```

Copy an array to the other y=x.

#### Parameters

<i>n</i>	Number of variables
<i>x</i>	Pointer to the original vector
<i>y</i>	Pointer to the destination vector

#### Author

Chunsheng Feng, Xiaoqiang Yue

#### Date

05/23/2012

Definition at line 184 of file AuxArray.c.

### 9.1.2.4 fasp\_iarray\_set()

```
void fasp_iarray_set (
    const INT n,
    INT * x,
    const INT val )
```

Set initial value for an array to be x=val.

#### Parameters

<i>n</i>	Number of variables
<i>x</i>	Pointer to the vector
<i>val</i>	Initial value for the REAL array

#### Author

Chensong Zhang

#### Date

04/03/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/25/2012  
Definition at line 103 of file AuxArray.c.

## 9.2 AuxConvert.c File Reference

Utilities for encoding format conversion.

```
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- unsigned long [fasp\\_aux\\_change\\_endian4](#) (const unsigned long x)  
*Swap order for different endian systems.*
- double [fasp\\_aux\\_change\\_endian8](#) (const double x)  
*Swap order for different endian systems.*
- double [fasp\\_aux\\_bbyteToldouble](#) (const unsigned char bytes[])  
*Swap order of double-precision float for different endian systems.*

### 9.2.1 Detailed Description

Utilities for encoding format conversion.

#### Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.2.2 Function Documentation

#### 9.2.2.1 [fasp\\_aux\\_bbyteToldouble\(\)](#)

```
double fasp_aux_bbyteToldouble (
    const unsigned char bytes[] )
```

Swap order of double-precision float for different endian systems.

#### Parameters

<i>bytes</i>	A unsigned char
--------------	-----------------

#### Returns

Unsigend long ineger after swapping

#### Author

Chensong Zhang

#### Date

11/16/2009

Definition at line 81 of file AuxConvert.c.

### 9.2.2.2 fasp\_aux\_change\_endian4()

```
unsigned long fasp_aux_change_endian4 (  
    const unsigned long x )
```

Swap order for different endian systems.

#### Parameters

<i>x</i>	An unsigned long integer
----------	--------------------------

#### Returns

Unsigned long integer after swapping

#### Author

Chensong Zhang

#### Date

11/16/2009

Definition at line 32 of file AuxConvert.c.

### 9.2.2.3 fasp\_aux\_change\_endian8()

```
double fasp_aux_change_endian8 (  
    const double x )
```

Swap order for different endian systems.

#### Parameters

<i>x</i>	A unsigned long integer
----------	-------------------------

#### Returns

Unsigned long integer after swapping

#### Author

Chensong Zhang

#### Date

11/16/2009

Definition at line 50 of file AuxConvert.c.

## 9.3 AuxGivens.c File Reference

Givens transformation.

```
#include <math.h>  
#include "fasp.h"  
#include "fasp_functs.h"
```

## Functions

- void `fasp_aux_givens` (const `REAL` beta, const `dCSRmat` \*H, `dvector` \*y, `REAL` \*work)  
Perform Givens rotations to compute  $y \mid \text{beta} * e_1 - H * y \mid$ .

### 9.3.1 Detailed Description

Givens transformation.

Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2008–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.3.2 Function Documentation

#### 9.3.2.1 `fasp_aux_givens()`

```
void fasp_aux_givens (
    const REAL beta,
    const dCSRmat * H,
    dvector * y,
    REAL * work )
```

Perform Givens rotations to compute  $y \mid \text{beta} * e_1 - H * y \mid$ .

Parameters

<i>beta</i>	Norm of residual $r_0$
<i>H</i>	Upper Hessenberg <code>dCSRmat</code> matrix: $(m+1)*m$
<i>y</i>	Minimizer of $\mid \text{beta} * e_1 - H * y \mid$
<i>work</i>	Temporary work array

Author

Xuehai Huang

Date

10/19/2008

Definition at line 36 of file AuxGivens.c.

## 9.4 AuxGraphics.c File Reference

Graphical output for CSR matrix.

```
#include <math.h>
#include "fasp.h"
#include "fasp_grid.h"
#include "fasp_functs.h"
```

## Functions

- void [fasp\\_dcsr\\_subplot](#) (const [dCSRmat](#) \*A, const char \*filename, int size)  
*Write sparse matrix pattern in BMP file format.*
- void [fasp\\_dcsr\\_plot](#) (const [dCSRmat](#) \*A, const char \*fname)  
*Write dCSR sparse matrix pattern in BMP file format.*
- void [fasp\\_dbsr\\_subplot](#) (const [dBSRmat](#) \*A, const char \*filename, int size)  
*Write sparse matrix pattern in BMP file format.*
- void [fasp\\_dbsr\\_plot](#) (const [dBSRmat](#) \*A, const char \*fname)  
*Write dBSR sparse matrix pattern in BMP file format.*
- void [fasp\\_grid2d\\_plot](#) ([pgrid2d](#) pg, int level)  
*Output grid to a EPS file.*

### 9.4.1 Detailed Description

Graphical output for CSR matrix.

#### Note

This file contains Level-0 (Aux) functions. It requires: [AuxMemory.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.4.2 Function Documentation

#### 9.4.2.1 [fasp\\_dbsr\\_plot\(\)](#)

```
void fasp_dbsr_plot (
    const dBSRmat * A,
    const char * fname )
```

Write dBSR sparse matrix pattern in BMP file format.

#### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>fname</i>	File name

#### Author

Chunsheng Feng

#### Date

11/16/2013

#### Note

The routine [fasp\\_dbsr\\_plot](#) writes pattern of the specified [dBSRmat](#) matrix in uncompressed BMP file format (Windows bitmap) to a binary file whose name is specified by the character string filename.

Each pixel corresponds to one matrix element. The pixel colors have the following meaning:

White structurally zero element Black zero element Blue positive element Red negative element Brown nearly zero element

Definition at line 339 of file AuxGraphics.c.

#### 9.4.2.2 fasp\_dbsr\_subplot()

```
void fasp_dbsr_subplot (
    const dBSRmat * A,
    const char * filename,
    int size )
```

Write sparse matrix pattern in BMP file format.

##### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>filename</i>	File name
<i>size</i>	size*size is the picture size for the picture

##### Author

Chunsheng Feng

##### Date

11/16/2013

##### Note

The routine `fasp_dbsr_subplot` writes pattern of the specified [dBSRmat](#) matrix in uncompressed BMP file format (Windows bitmap) to a binary file whose name is specified by the character string `filename`.

Each pixel corresponds to one matrix element. The pixel colors have the following meaning:

White structurally zero element Black zero element Blue positive element Red negative element Brown nearly zero element

Definition at line 259 of file AuxGraphics.c.

#### 9.4.2.3 fasp\_dcsr\_plot()

```
void fasp_dcsr_plot (
    const dCSRmat * A,
    const char * fname )
```

Write dCSR sparse matrix pattern in BMP file format.

##### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>fname</i>	File name to plot to

##### Author

Chunsheng Feng



**Date**

11/16/2013

**Note**

The routine `fasp_dcsr_plot` writes pattern of the specified `dCSRmat` matrix in uncompressed BMP file format (Windows bitmap) to a binary file whose name is specified by the character string `filename`.

Each pixel corresponds to one matrix element. The pixel colors have the following meaning:

White structurally zero element Black zero element Blue positive element Red negative element Brown nearly zero element

Definition at line 117 of file `AuxGraphics.c`.

**9.4.2.4 fasp\_dcsr\_subplot()**

```
void fasp_dcsr_subplot (
    const dCSRmat * A,
    const char * filename,
    int size )
```

Write sparse matrix pattern in BMP file format.

**Parameters**

<i>A</i>	Pointer to the <code>dCSRmat</code> matrix
<i>filename</i>	File name
<i>size</i>	<code>size*size</code> is the picture size for the picture

**Author**

Chensong Zhang

**Date**

03/29/2009

**Note**

The routine `fasp_dcsr_subplot` writes pattern of the specified `dCSRmat` matrix in uncompressed BMP file format (Windows bitmap) to a binary file whose name is specified by the character string `filename`.

Each pixel corresponds to one matrix element. The pixel colors have the following meaning:

White structurally zero element Blue positive element Red negative element Brown nearly zero element

Definition at line 57 of file `AuxGraphics.c`.

**9.4.2.5 fasp\_grid2d\_plot()**

```
void fasp_grid2d_plot (
    pgrid2d pg,
    int level )
```

Output grid to a EPS file.

## Parameters

<i>pg</i>	Pointer to grid in 2d
<i>level</i>	Number of levels

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 478 of file AuxGraphics.c.

## 9.5 AuxInput.c File Reference

Read and check input parameters.

```
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [SHORT fasp\\_param\\_check](#) ([input\\_param](#) \*inparam)  
*Simple check on input parameters.*
- void [fasp\\_param\\_input](#) (const char \*fname, [input\\_param](#) \*inparam)  
*Read input parameters from disk file.*

#### 9.5.1 Detailed Description

Read and check input parameters.

## Note

This file contains Level-0 (Aux) functions. It requires: [AuxMemory.c](#) and [AuxMessage.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.5.2 Function Documentation

##### 9.5.2.1 fasp\_param\_check()

```
SHORT fasp_param_check (
    input_param * inparam )
```

Simple check on input parameters.

## Parameters

<i>inparam</i>	Input parameters
----------------	------------------

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Chensong Zhang

**Date**

09/29/2013

Definition at line 33 of file AuxInput.c.

**9.5.2.2 fasp\_param\_input()**

```
void fasp_param_input (
    const char * fname,
    input_param * inparam )
```

Read input parameters from disk file.

**Parameters**

<i>fname</i>	File name for input file
<i>inparam</i>	Input parameters

**Author**

Chensong Zhang

**Date**

03/20/2010

Modified by Xiaozhe Hu on 01/23/2011: add AMLI cycle; Modified by Chensong Zhang on 05/10/2013: add a new input; Modified by Chensong Zhang on 03/23/2015: skip unknown keyword; Modified by Chensong Zhang on 03/27/2017: check unexpected error; Modified by Chensong Zhang on 09/20/2017: new skip the line;  
Definition at line 112 of file AuxInput.c.

**9.6 AuxMemory.c File Reference**

Memory allocation and deallocation subroutines.

```
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void \* **fasp\_mem\_malloc** (const unsigned int size, const unsigned int type)
- void \* **fasp\_mem\_realloc** (void \*oldmem, const **LONGLONG** tsize)  
*Reallocate, initiate, and check memory.*
- void **fasp\_mem\_free** (void \*mem)  
*Free up previous allocated memory body and set pointer to NULL.*

- void `fasp_mem_usage` (void)  
*Show total allocated memory currently.*
- `SHORT fasp_mem_iludata_check` (const `ILU_data` \*iludata)  
*Check whether a `ILU_data` has enough work space.*

## Variables

- const int `Million` = 1048576

### 9.6.1 Detailed Description

Memory allocation and deallocation subroutines.

#### Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.6.2 Function Documentation

#### 9.6.2.1 `fasp_mem_free()`

```
void fasp_mem_free (
    void * mem )
```

Free up previous allocated memory body and set pointer to NULL.

#### Parameters

<code>mem</code>	Pointer to the memory body need to be freed
------------------	---

#### Author

Chensong Zhang

#### Date

2010/12/24

Modified on 2018/01/10 by Chensong: Add output when mem is NULL  
Definition at line 155 of file AuxMemory.c.

#### 9.6.2.2 `fasp_mem_iludata_check()`

```
SHORT fasp_mem_iludata_check (
    const ILU_data * iludata )
```

Check whether a `ILU_data` has enough work space.

#### Parameters

<code>iludata</code>	Pointer to be checked
----------------------	-----------------------

**Returns**

FASP\_SUCCESS if success, else ERROR (negative value)

**Author**

Xiaozhe Hu, Chensong Zhang

**Date**

11/27/09

Definition at line 205 of file AuxMemory.c.

**9.6.2.3 fasp\_mem\_realloc()**

```
void * fasp_mem_realloc (
    void * oldmem,
    const LONGLONG tsize )
```

Reallocate, initiate, and check memory.

**Parameters**

<i>oldmem</i>	Pointer to the existing mem block
<i>tsize</i>	Size of memory blocks

**Returns**

Void pointer to the reallocated memory

**Author**

Chensong Zhang

**Date**

2010/08/12

Modified by Chensong Zhang on 07/30/2013: print error if failed

Definition at line 114 of file AuxMemory.c.

**9.6.2.4 fasp\_mem\_usage()**

```
void fasp_mem_usage (
    void )
```

Show total allocated memory currently.

**Author**

Chensong Zhang

**Date**

2010/08/12

Definition at line 185 of file AuxMemory.c.

## 9.7 AuxMessage.c File Reference

Output some useful messages.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void `fasp_itinfo` (const `INT` `ptlvl`, const `INT` `stop_type`, const `INT` `iter`, const `REAL` `relres`, const `REAL` `absres`, const `REAL` `factor`)  
*Print out iteration information for iterative solvers.*
- void `fasp_amgcomplexity` (const `AMG_data` `*mgl`, const `SHORT` `ptlvl`)  
*Print level and complexity information of AMG.*
- void `fasp_amgcomplexity_bsr` (const `AMG_data_bsr` `*mgl`, const `SHORT` `ptlvl`)  
*Print complexities of AMG method for BSR matrices.*
- void `fasp_cputime` (const char `*message`, const `REAL` `cputime`)  
*Print CPU walltime.*
- void `fasp_message` (const `INT` `ptlvl`, const char `*message`)  
*Print output information if necessary.*
- void `fasp_chkerr` (const `SHORT` `status`, const char `*fctname`)  
*Check error status and print out error messages before quit.*

### 9.7.1 Detailed Description

Output some useful messages.

#### Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.7.2 Function Documentation

#### 9.7.2.1 `fasp_amgcomplexity()`

```
void void fasp_amgcomplexity (
    const AMG_data * mgl,
    const SHORT ptlvl )
```

Print level and complexity information of AMG.

#### Parameters

<i>mgl</i>	Multilevel hierachy for AMG
<i>ptlvl</i>	How much information to print

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 84 of file AuxMessage.c.

**9.7.2.2 fasp\_amgcomplexity\_bsr()**

```
void void fasp_amgcomplexity_bsr (
    const AMG_data_bsr * mgl,
    const SHORT prtlvl )
```

Print complexities of AMG method for BSR matrices.

**Parameters**

<i>mgl</i>	Multilevel hierachy for AMG
<i>prtlvl</i>	How much information to print

**Author**

Chensong Zhang

**Date**

05/10/2013

Definition at line 136 of file AuxMessage.c.

**9.7.2.3 fasp\_chkerr()**

```
void fasp_chkerr (
    const SHORT status,
    const char * fctname )
```

Check error status and print out error messages before quit.

**Parameters**

<i>status</i>	Error status
<i>fctname</i>	Function name where this routine is called

**Author**

Chensong Zhang

**Date**

01/10/2012

Definition at line 213 of file AuxMessage.c.

#### 9.7.2.4 fasp\_cputime()

```
void void fasp_cputime (
    const char * message,
    const REAL cputime )
```

Print CPU walltime.

##### Parameters

<i>message</i>	Some string to print out
<i>cputime</i>	Walltime since start to end

##### Author

Chensong Zhang

##### Date

04/10/2012

Definition at line 179 of file AuxMessage.c.

#### 9.7.2.5 fasp\_itinfo()

```
void fasp_itinfo (
    const INT ptrlvl,
    const INT stop_type,
    const INT iter,
    const REAL relres,
    const REAL absres,
    const REAL factor )
```

Print out iteration information for iterative solvers.

##### Parameters

<i>ptrlvl</i>	Level for output
<i>stop_type</i>	Type of stopping criteria
<i>iter</i>	Number of iterations
<i>relres</i>	Relative residual of different kinds
<i>absres</i>	Absolute residual of different kinds
<i>factor</i>	Contraction factor

##### Author

Chensong Zhang

##### Date

11/16/2009

Modified by Chensong Zhang on 03/28/2013: Output initial guess Modified by Chensong Zhang on 04/05/2013: Fix a typo

Definition at line 41 of file AuxMessage.c.



### 9.7.2.6 fasp\_message()

```
void fasp_message (
    const INT ptrlvl,
    const char * message )
```

Print output information if necessary.

#### Parameters

<i>ptrlvl</i>	Level for output
<i>message</i>	Error message to print

#### Author

Chensong Zhang

#### Date

11/16/2009

Definition at line 196 of file AuxMessage.c.

## 9.8 AuxParam.c File Reference

Initialize, set, or print input data and parameters.

```
#include <stdio.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_param\\_set](#) (const int argc, const char \*argv[], [input\\_param](#) \*iniparam)  
*Read input from command-line arguments.*
- void [fasp\\_param\\_init](#) (const [input\\_param](#) \*iniparam, [ITS\\_param](#) \*itsparam, [AMG\\_param](#) \*amgparam, [ILU\\_param](#) \*iluparam, [SWZ\\_param](#) \*swzparam)  
*Initialize parameters, global variables, etc.*
- void [fasp\\_param\\_input\\_init](#) ([input\\_param](#) \*iniparam)  
*Initialize input parameters.*
- void [fasp\\_param\\_amg\\_init](#) ([AMG\\_param](#) \*amgparam)  
*Initialize AMG parameters.*
- void [fasp\\_param\\_solver\\_init](#) ([ITS\\_param](#) \*itsparam)  
*Initialize ITS\_param.*
- void [fasp\\_param\\_ilu\\_init](#) ([ILU\\_param](#) \*iluparam)  
*Initialize ILU parameters.*
- void [fasp\\_param\\_swz\\_init](#) ([SWZ\\_param](#) \*swzparam)  
*Initialize Schwarz parameters.*
- void [fasp\\_param\\_amg\\_set](#) ([AMG\\_param](#) \*param, const [input\\_param](#) \*iniparam)  
*Set AMG\_param from INPUT.*
- void [fasp\\_param\\_ilu\\_set](#) ([ILU\\_param](#) \*iluparam, const [input\\_param](#) \*iniparam)  
*Set ILU\_param with INPUT.*

- void `fasp_param_swz_set` (`SWZ_param` \*swzparam, const `input_param` \*iniparam)  
*Set `SWZ_param` with `INPUT`.*
- void `fasp_param_solver_set` (`ITS_param` \*itsparam, const `input_param` \*iniparam)  
*Set `ITS_param` with `INPUT`.*
- void `fasp_param_amg_to_prec` (`precond_data` \*pcdata, const `AMG_param` \*amgparam)  
*Set `precond_data` with `AMG_param`.*
- void `fasp_param_prec_to_amg` (`AMG_param` \*amgparam, const `precond_data` \*pcdata)  
*Set `AMG_param` with `precond_data`.*
- void `fasp_param_amg_to_precbsr` (`precond_data_bsr` \*pcdata, const `AMG_param` \*amgparam)  
*Set `precond_data_bsr` with `AMG_param`.*
- void `fasp_param_precbsr_to_amg` (`AMG_param` \*amgparam, const `precond_data_bsr` \*pcdata)  
*Set `AMG_param` with `precond_data`.*
- void `fasp_param_amg_print` (const `AMG_param` \*param)  
*Print out AMG parameters.*
- void `fasp_param_ilu_print` (const `ILU_param` \*param)  
*Print out ILU parameters.*
- void `fasp_param_swz_print` (const `SWZ_param` \*param)  
*Print out Schwarz parameters.*
- void `fasp_param_solver_print` (const `ITS_param` \*param)  
*Print out itsolver parameters.*

### 9.8.1 Detailed Description

Initialize, set, or print input data and parameters.

Note

This file contains Level-0 (Aux) functions. It requires: [AuxInput.c](#) and [AuxMessage.c](#)

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.8.2 Function Documentation

#### 9.8.2.1 `fasp_param_amg_init()`

```
void fasp_param_amg_init (
    AMG_param * amgparam )
```

Initialize AMG parameters.

Parameters

<code>amgparam</code>	Parameters for AMG
-----------------------	--------------------

Author

Chensong Zhang

## Date

2010/04/03

Definition at line 407 of file AuxParam.c.

**9.8.2.2 fasp\_param\_amg\_print()**

```
void fasp_param_amg_print (
    const AMG_param * param )
```

Print out AMG parameters.

## Parameters

<i>param</i>	Parameters for AMG
--------------	--------------------

## Author

Chensong Zhang

## Date

2010/03/22

Definition at line 820 of file AuxParam.c.

**9.8.2.3 fasp\_param\_amg\_set()**

```
void fasp_param_amg_set (
    AMG_param * param,
    const input_param * iniparam )
```

Set [AMG\\_param](#) from INPUT.

## Parameters

<i>param</i>	Parameters for AMG
<i>iniparam</i>	Input parameters

## Author

Chensong Zhang

## Date

2010/03/23

Definition at line 537 of file AuxParam.c.

**9.8.2.4 fasp\_param\_amg\_to\_prec()**

```
void fasp_param_amg_to_prec (
    precondition_data * pcddata,
    const AMG_param * amgparam )
```

Set `precond_data` with `AMG_param`.

## Parameters

<i>pcdata</i>	Preconditioning data structure
<i>amgparam</i>	Parameters for AMG

## Author

Chensong Zhang

## Date

2011/01/10

Definition at line 687 of file AuxParam.c.

**9.8.2.5 fasp\_param\_amg\_to\_precbsr()**

```
void fasp_param_amg_to_precbsr (
    precondition_data_bsr * pcdata,
    const AMG_param * amgparam )
```

Set [precond\\_data\\_bsr](#) with [AMG\\_param](#).

## Parameters

<i>pcdata</i>	Preconditioning data structure
<i>amgparam</i>	Parameters for AMG

## Author

Xiaozhe Hu

## Date

02/06/2012

Definition at line 755 of file AuxParam.c.

**9.8.2.6 fasp\_param\_ilu\_init()**

```
void fasp_param_ilu_init (
    ILU_param * iluparam )
```

Initialize ILU parameters.

## Parameters

<i>iluparam</i>	Parameters for ILU
-----------------	--------------------

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 495 of file AuxParam.c.

**9.8.2.7 fasp\_param\_ilu\_print()**

```
void fasp_param_ilu_print (
    const ILU_param * param )
```

Print out ILU parameters.

## Parameters

<i>param</i>	Parameters for ILU
--------------	--------------------

## Author

Chensong Zhang

## Date

2011/12/20

Definition at line 943 of file AuxParam.c.

**9.8.2.8 fasp\_param\_ilu\_set()**

```
void fasp_param_ilu_set (
    ILU_param * iluparam,
    const input_param * iniparam )
```

Set *ILU\_param* with INPUT.

## Parameters

<i>iluparam</i>	Parameters for ILU
<i>iniparam</i>	Input parameters

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 612 of file AuxParam.c.

**9.8.2.9 fasp\_param\_init()**

```
void fasp_param_init (
    const input_param * iniparam,
    ITS_param * itsparam,
```

```
    AMG_param * amgparam,  
    ILU_param * iluparam,  
    SWZ_param * swzparam )
```

Initialize parameters, global variables, etc.

#### Parameters

<i>iniparam</i>	Input parameters
<i>itsparam</i>	Iterative solver parameters
<i>amgparam</i>	AMG parameters
<i>iluparam</i>	ILU parameters
<i>swzparam</i>	Schwarz parameters

#### Author

Chensong Zhang

#### Date

2010/08/12

Modified by Chensong Zhang (12/29/2013): rewritten  
Definition at line 283 of file AuxParam.c.

#### 9.8.2.10 fasp\_param\_input\_init()

```
void fasp_param_input_init (  
    input_param * iniparam )
```

Initialize input parameters.

#### Parameters

<i>iniparam</i>	Input parameters
-----------------	------------------

#### Author

Chensong Zhang

#### Date

2010/03/20

Definition at line 325 of file AuxParam.c.

#### 9.8.2.11 fasp\_param\_prec\_to\_amg()

```
void fasp_param_prec_to_amg (  
    AMG_param * amgparam,  
    const precondition_data * pcd_data )
```

Set [AMG\\_param](#) with [precond\\_data](#).

## Parameters

<i>amgparam</i>	Parameters for AMG
<i>pcdata</i>	Preconditioning data structure

## Author

Chensong Zhang

## Date

2011/01/10

Definition at line 722 of file AuxParam.c.

**9.8.2.12 fasp\_param\_precbsr\_to\_amg()**

```
void fasp_param_precbsr_to_amg (
    AMG_param * amgparam,
    const precondition_data_bsr * pcdata )
```

Set [AMG\\_param](#) with [precond\\_data](#).

## Parameters

<i>amgparam</i>	Parameters for AMG
<i>pcdata</i>	Preconditioning data structure

## Author

Xiaozhe Hu

## Date

02/06/2012

Definition at line 790 of file AuxParam.c.

**9.8.2.13 fasp\_param\_set()**

```
void fasp_param_set (
    const int argc,
    const char * argv[],
    input_param * iniparam )
```

Read input from command-line arguments.

## Parameters

<i>argc</i>	Number of arg input
<i>argv</i>	Input arguments
<i>iniparam</i>	Parameters to be set



## Author

Chensong Zhang

## Date

12/29/2013

Definition at line 41 of file AuxParam.c.

**9.8.2.14 fasp\_param\_solver\_init()**

```
void fasp_param_solver_init (
    ITS_param * itsparam )
```

Initialize ITS\_param.

## Parameters

<i>itsparam</i>	Parameters for iterative solvers
-----------------	----------------------------------

## Author

Chensong Zhang

## Date

2010/03/23

Definition at line 473 of file AuxParam.c.

**9.8.2.15 fasp\_param\_solver\_print()**

```
void fasp_param_solver_print (
    const ITS_param * param )
```

Print out itsolver parameters.

## Parameters

<i>param</i>	Paramters for iterative solvers
--------------	---------------------------------

## Author

Chensong Zhang

## Date

2011/12/20

Definition at line 1002 of file AuxParam.c.

**9.8.2.16 fasp\_param\_solver\_set()**

```
void fasp_param_solver_set (
```

```
ITS_param * itsparam,
const input_param * iniparam )
```

Set `ITS_param` with INPUT.

#### Parameters

<i>itsparam</i>	Parameters for iterative solvers
<i>iniparam</i>	Input parameters

#### Author

Chensong Zhang

#### Date

2010/03/23

Definition at line 656 of file AuxParam.c.

### 9.8.2.17 fasp\_param\_swz\_init()

```
void fasp_param_swz_init (
    SWZ_param * swzparam )
```

Initialize Schwarz parameters.

#### Parameters

<i>swzparam</i>	Parameters for Schwarz method
-----------------	-------------------------------

#### Author

Xiaozhe Hu

#### Date

05/22/2012

Modified by Chensong Zhang on 10/10/2014: Add block solver type

Definition at line 517 of file AuxParam.c.

### 9.8.2.18 fasp\_param\_swz\_print()

```
void fasp_param_swz_print (
    const SWZ_param * param )
```

Print out Schwarz parameters.

#### Parameters

<i>param</i>	Parameters for Schwarz
--------------	------------------------

## Author

Xiaozhe Hu

## Date

05/22/2012

Definition at line 973 of file AuxParam.c.

### 9.8.2.19 fasp\_param\_swz\_set()

```
void fasp_param_swz_set (
    SWZ_param * swzparam,
    const input_param * iniparam )
```

Set [SWZ\\_param](#) with INPUT.

## Parameters

<i>swzparam</i>	Parameters for Schwarz method
<i>iniparam</i>	Input parameters

## Author

Xiaozhe Hu

## Date

05/22/2012

Definition at line 634 of file AuxParam.c.

## 9.9 AuxSort.c File Reference

Array sorting/merging and removing duplicated integers.

```
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [INT fasp\\_aux\\_BiSearch](#) (const [INT](#) nlist, const [INT](#) \*list, const [INT](#) value)  
*Binary Search.*
- [INT fasp\\_aux\\_unique](#) ([INT](#) numbers[], const [INT](#) size)  
*Remove duplicates in an sorted (ascending order) array.*
- void [fasp\\_aux\\_merge](#) ([INT](#) numbers[], [INT](#) work[], [INT](#) left, [INT](#) mid, [INT](#) right)  
*Merge two sorted arrays.*
- void [fasp\\_aux\\_msort](#) ([INT](#) numbers[], [INT](#) work[], [INT](#) left, [INT](#) right)  
*Sort the INT array in ascending order with the merge sort algorithm.*
- void [fasp\\_aux\\_iQuickSort](#) ([INT](#) \*a, [INT](#) left, [INT](#) right)  
*Sort the array (INT type) in ascending order with the quick sorting algorithm.*
- void [fasp\\_aux\\_dQuickSort](#) ([REAL](#) \*a, [INT](#) left, [INT](#) right)

*Sort the array (REAL type) in ascending order with the quick sorting algorithm.*

- void `fasp_aux_iQuickSortIndex` (INT \*a, INT left, INT right, INT \*index)

*Reorder the index of (INT type) so that 'a' is in ascending order.*

- void `fasp_aux_dQuickSortIndex` (REAL \*a, INT left, INT right, INT \*index)

*Reorder the index of (REAL type) so that 'a' is ascending in such order.*

### 9.9.1 Detailed Description

Array sorting/merging and removing duplicated integers.

Note

This file contains Level-0 (Aux) functions. It requires: [AuxMemory.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.9.2 Function Documentation

#### 9.9.2.1 fasp\_aux\_BiSearch()

```
INT fasp_aux_BiSearch (
    const INT nlist,
    const INT * list,
    const INT value )
```

Binary Search.

Parameters

<i>nlist</i>	Length of the array list
<i>list</i>	Pointer to a set of values
<i>value</i>	The target

Returns

The location of value in array list if succeeded; otherwise, return -1.

Author

Chunsheng Feng

Date

03/01/2011

Definition at line 42 of file AuxSort.c.

#### 9.9.2.2 fasp\_aux\_dQuickSort()

```
void fasp_aux_dQuickSort (
    REAL * a,
```

```
    INT left,  
    INT right )
```

Sort the array (REAL type) in ascending order with the quick sorting algorithm.

**Parameters**

<i>a</i>	Pointer to the array needed to be sorted
<i>left</i>	Starting index
<i>right</i>	Ending index

**Author**

Zhiyang Zhou

**Date**

2009/11/28

**Note**

'left' and 'right' are usually set to be 0 and n-1, respectively where n is the length of 'a'.

Definition at line 246 of file AuxSort.c.

**9.9.2.3 fasp\_aux\_dQuickSortIndex()**

```
void fasp_aux_dQuickSortIndex (
    REAL * a,
    INT left,
    INT right,
    INT * index )
```

Reorder the index of (REAL type) so that 'a' is ascending in such order.

**Parameters**

<i>a</i>	Pointer to the array
<i>left</i>	Starting index
<i>right</i>	Ending index
<i>index</i>	Index of 'a' (out)

**Author**

Zhiyang Zhou

**Date**

2009/12/02

**Note**

'left' and 'right' are usually set to be 0 and n-1, respectively, where n is the length of 'a'. 'index' should be initialized in the nature order and it has the same length as 'a'.

Definition at line 327 of file AuxSort.c.

#### 9.9.2.4 fasp\_aux\_iQuickSort()

```
void fasp_aux_iQuickSort (
    INT * a,
    INT left,
    INT right )
```

Sort the array (INT type) in ascending order with the quick sorting algorithm.

##### Parameters

<i>a</i>	Pointer to the array needed to be sorted
<i>left</i>	Starting index
<i>right</i>	Ending index

##### Author

Zhiyang Zhou

##### Date

11/28/2009

##### Note

'left' and 'right' are usually set to be 0 and n-1, respectively where n is the length of 'a'.

Definition at line 208 of file AuxSort.c.

#### 9.9.2.5 fasp\_aux\_iQuickSortIndex()

```
void fasp_aux_iQuickSortIndex (
    INT * a,
    INT left,
    INT right,
    INT * index )
```

Reorder the index of (INT type) so that 'a' is in ascending order.

##### Parameters

<i>a</i>	Pointer to the array
<i>left</i>	Starting index
<i>right</i>	Ending index
<i>index</i>	Index of 'a' (out)

##### Author

Zhiyang Zhou

##### Date

2009/12/02

**Note**

'left' and 'right' are usually set to be 0 and n-1, respectively, where n is the length of 'a'. 'index' should be initialized in the nature order and it has the same length as 'a'.

Definition at line 286 of file AuxSort.c.

**9.9.2.6 fasp\_aux\_merge()**

```
void fasp_aux_merge (
    INT numbers[],
    INT work[],
    INT left,
    INT mid,
    INT right )
```

Merge two sorted arrays.

**Parameters**

<i>numbers</i>	Pointer to the array needed to be sorted
<i>work</i>	Pointer to the work array with same size as numbers
<i>left</i>	Starting index of array 1
<i>mid</i>	Starting index of array 2
<i>right</i>	Ending index of array 1 and 2

**Author**

Chensong Zhang

**Date**

11/21/2010

**Note**

Both arrays are stored in numbers! Arrays should be pre-sorted!

Definition at line 115 of file AuxSort.c.

**9.9.2.7 fasp\_aux\_msort()**

```
void fasp_aux_msort (
    INT numbers[],
    INT work[],
    INT left,
    INT right )
```

Sort the INT array in ascending order with the merge sort algorithm.

**Parameters**

<i>numbers</i>	Pointer to the array needed to be sorted
<i>work</i>	Pointer to the work array with same size as numbers
<i>left</i>	Starting index
<i>right</i>	Ending index



**Author**

Chensong Zhang

**Date**

11/21/2010

**Note**

'left' and 'right' are usually set to be 0 and n-1, respectively

Definition at line 177 of file AuxSort.c.

**9.9.2.8 fasp\_aux\_unique()**

```
INT fasp_aux_unique (
    INT numbers[ ],
    const INT size )
```

Remove duplicates in an sorted (ascending order) array.

**Parameters**

<i>numbers</i>	Pointer to the array needed to be sorted (in/out)
<i>size</i>	Length of the target array

**Returns**

New size after removing duplicates

**Author**

Chensong Zhang

**Date**

11/21/2010

**Note**

Operation is in place. Does not use any extra or temporary storage.

Definition at line 82 of file AuxSort.c.

## 9.10 AuxThreads.c File Reference

Get and set number of threads and assign work load for each thread.

```
#include <stdio.h>
#include <stdlib.h>
#include "fasp.h"
```

## Functions

- void `fasp_get_start_end` (const `INT` `procid`, const `INT` `nprocs`, const `INT` `n`, `INT` `*start`, `INT` `*end`)  
*Assign Load to each thread.*
- void `fasp_set_gs_threads` (const `INT` `mythreads`, const `INT` `its`)  
*Set threads for CPR. Please add it at the begin of Krylov OpenMP method function and after `iter++`.*

## Variables

- `INT` `THDs_AMG_GS` =0
- `INT` `THDs_CPR_IGS` =0
- `INT` `THDs_CPR_gGS` =0

### 9.10.1 Detailed Description

Get and set number of threads and assign work load for each thread.

#### Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.10.2 Function Documentation

#### 9.10.2.1 `fasp_get_start_end()`

```
void fasp_get_start_end (
    const INT procid,
    const INT nprocs,
    const INT n,
    INT * start,
    INT * end )
```

Assign Load to each thread.

#### Parameters

<code>procid</code>	Index of thread
<code>nprocs</code>	Number of threads
<code>n</code>	Total workload
<code>start</code>	Pointer to the begin of each thread in total workload
<code>end</code>	Pointer to the end of each thread in total workload

#### Author

Chunsheng Feng, Xiaoqiang Yue and Zheng Li

#### Date

June/25/2012

Definition at line 92 of file AuxThreads.c.

### 9.10.2.2 fasp\_set\_gs\_threads()

```
void fasp_set_gs_threads (
    const INT mythreads,
    const INT its )
```

Set threads for CPR. Please add it at the begin of Krylov OpenMP method function and after iter++.

#### Parameters

<i>mythreads</i>	Total threads of solver
<i>its</i>	Current iteration number in the Krylov methods

#### Author

Feng Chunsheng, Yue Xiaoqiang

#### Date

03/20/2011

Definition at line 132 of file AuxThreads.c.

## 9.10.3 Variable Documentation

### 9.10.3.1 THDs\_AMG\_GS

```
INT THDs_AMG_GS =0
```

AMG GS smoothing threads

Definition at line 116 of file AuxThreads.c.

### 9.10.3.2 THDs\_CPR\_gGS

```
INT THDs_CPR_gGS =0
```

global matrix GS smoothing threads

Definition at line 118 of file AuxThreads.c.

### 9.10.3.3 THDs\_CPR\_lGS

```
INT THDs_CPR_lGS =0
```

reservoir GS smoothing threads

Definition at line 117 of file AuxThreads.c.

## 9.11 AuxTiming.c File Reference

Timing subroutines.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- void `fasp_gettime` (`REAL *time`)  
*Get system time.*

### 9.11.1 Detailed Description

Timing subroutines.

#### Note

This file contains Level-0 (Aux) functions.

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.11.2 Function Documentation

#### 9.11.2.1 `fasp_gettime()`

```
void fasp_gettime (
    REAL * time )
```

Get system time.

#### Author

Chunsheng Feng, Zheng LI

#### Date

11/10/2012

Modified by Chensong Zhang on 09/22/2014: Use CLOCKS\_PER\_SEC for cross-platform Definition at line 36 of file AuxTiming.c.

## 9.12 AuxVector.c File Reference

Simple vector operations – init, set, copy, etc.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- `SHORT fasp_dvec_isnan` (const `dvector *u`)  
*Check a dvector whether there is NAN.*
- `dvector fasp_dvec_create` (const `INT m`)  
*Create dvector data space of REAL type.*
- `ivector fasp_ivec_create` (const `INT m`)  
*Create vector data space of INT type.*
- void `fasp_dvec_alloc` (const `INT m`, `dvector *u`)  
*Create dvector data space of REAL type.*

- void `fasp_ivec_alloc` (const `INT` `m`, `ivector` `*u`)  
*Create vector data space of `INT` type.*
- void `fasp_dvec_free` (`dvector` `*u`)  
*Free vector data space of `REAL` type.*
- void `fasp_ivec_free` (`ivector` `*u`)  
*Free vector data space of `INT` type.*
- void `fasp_dvec_rand` (const `INT` `n`, `dvector` `*x`)  
*Generate fake random `REAL` vector in the range from 0 to 1.*
- void `fasp_dvec_set` (`INT` `n`, `dvector` `*x`, const `REAL` `val`)  
*Initialize `dvector` `x[i]=val` for `i=0:n-1`.*
- void `fasp_ivec_set` (`INT` `n`, `ivector` `*u`, const `INT` `m`)  
*Set `ivector` value to be `m`.*
- void `fasp_dvec_cp` (const `dvector` `*x`, `dvector` `*y`)  
*Copy `dvector` `x` to `dvector` `y`.*
- `REAL` `fasp_dvec_maxdiff` (const `dvector` `*x`, const `dvector` `*y`)  
*Maximal difference of two `dvector` `x` and `y`.*
- void `fasp_dvec_symdiagscale` (`dvector` `*b`, const `dvector` `*diag`)  
*Symmetric diagonal scaling  $D^{-1/2}b$ .*

### 9.12.1 Detailed Description

Simple vector operations – init, set, copy, etc.

#### Note

This file contains Level-0 (Aux) functions. It requires: [AuxThreads.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.12.2 Function Documentation

#### 9.12.2.1 `fasp_dvec_alloc()`

```
void fasp_dvec_alloc (
    const INT m,
    dvector *u )
```

Create `dvector` data space of `REAL` type.

#### Parameters

<code>m</code>	Number of rows
<code>u</code>	Pointer to <code>dvector</code> (OUTPUT)

#### Author

Chensong Zhang

**Date**

2010/04/06

Definition at line 105 of file AuxVector.c.

**9.12.2.2 fasp\_dvec\_cp()**

```
void fasp_dvec_cp (
    const dvector * x,
    dvector * y )
```

Copy dvector x to dvector y.

**Parameters**

<i>x</i>	Pointer to dvector
<i>y</i>	Pointer to dvector (MODIFIED)

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 334 of file AuxVector.c.

**9.12.2.3 fasp\_dvec\_create()**

```
dvector fasp_dvec_create (
    const INT m )
```

Create dvector data space of REAL type.

**Parameters**

<i>m</i>	Number of rows
----------	----------------

**Returns**

u The new dvector

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 62 of file AuxVector.c.

#### 9.12.2.4 fasp\_dvec\_free()

```
void fasp_dvec_free (
    dvector * u )
```

Free vector data space of REAL type.

##### Parameters

<i>u</i>	Pointer to dvector which needs to be deallocated
----------	--

##### Author

Chensong Zhang

##### Date

2010/04/03

Definition at line 145 of file AuxVector.c.

#### 9.12.2.5 fasp\_dvec\_isnan()

```
SHORT fasp_dvec_isnan (
    const dvector * u )
```

Check a dvector whether there is NAN.

##### Parameters

<i>u</i>	Pointer to dvector
----------	--------------------

##### Returns

Return TRUE if there is NAN

##### Author

Chensong Zhang

##### Date

2013/03/31

Definition at line 39 of file AuxVector.c.

#### 9.12.2.6 fasp\_dvec\_maxdiff()

```
REAL fasp_dvec_maxdiff (
    const dvector * x,
    const dvector * y )
```

Maximal difference of two dvector x and y.

**Parameters**

<i>x</i>	Pointer to dvector
<i>y</i>	Pointer to dvector

**Returns**

Maximal norm of x-y

**Author**

Chensong Zhang

**Date**

11/16/2009

Modified by chunsheng Feng, Zheng Li

**Date**

06/30/2012

Definition at line 357 of file AuxVector.c.

**9.12.2.7 fasp\_dvec\_rand()**

```
void fasp_dvec_rand (
    const INT n,
    dvector * x )
```

Generate fake random REAL vector in the range from 0 to 1.

**Parameters**

<i>n</i>	Size of the vector
<i>x</i>	Pointer to dvector

**Note**

Sample usage:

```
dvector xapp;
```

```
fasp_dvec_create(100,&xapp);
```

```
fasp_dvec_rand(100,&xapp);
```

```
fasp_dvec_print(100,&xapp);
```



**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 192 of file AuxVector.c.

**9.12.2.8 fasp\_dvec\_set()**

```
void fasp_dvec_set (
    INT n,
    dvector * x,
    const REAL val )
```

Initialize dvector  $x[i]=val$  for  $i=0:n-1$ .**Parameters**

$n$	Number of variables
$x$	Pointer to dvector
$val$	Initial value for the vector

**Author**

Chensong Zhang

**Date**

11/16/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

Definition at line 222 of file AuxVector.c.

**9.12.2.9 fasp\_dvec\_symdiagscale()**

```
void fasp_dvec_symdiagscale (
    dvector * b,
    const dvector * diag )
```

Symmetric diagonal scaling  $D^{-1/2}b$ .**Parameters**

$b$	Pointer to dvector
$diag$	Pointer to dvector: the diagonal entries

**Author**

Xiaozhe Hu

**Date**

01/31/2011

Definition at line 410 of file AuxVector.c.

**9.12.2.10 fasp\_ivec\_alloc()**

```
void fasp_ivec_alloc (  
    const INT m,  
    ivector * u )
```

Create vector data space of INT type.

**Parameters**

<i>m</i>	Number of rows
<i>u</i>	Pointer to ivector (OUTPUT)

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 125 of file AuxVector.c.

**9.12.2.11 fasp\_ivec\_create()**

```
ivector fasp_ivec_create (  
    const INT m )
```

Create vector data space of INT type.

**Parameters**

<i>m</i>	Number of rows
----------	----------------

**Returns**

u The new ivector

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 84 of file AuxVector.c.

**9.12.2.12 fasp\_ivec\_free()**

```
void fasp_ivec_free (
    ivector * u )
```

Free vector data space of INT type.

**Parameters**

<i>u</i>	Pointer to ivector which needs to be deallocated
----------	--

**Author**

Chensong Zhang

**Date**

2010/04/03

**Note**

This function is same as fasp\_dvec\_free except input type.

Definition at line 164 of file AuxVector.c.

**9.12.2.13 fasp\_ivec\_set()**

```
void fasp_ivec_set (
    INT n,
    ivector * u,
    const INT m )
```

Set ivector value to be m.

**Parameters**

<i>n</i>	Number of variables
<i>m</i>	Integer value of ivector
<i>u</i>	Pointer to ivector (MODIFIED)

**Author**

Chensong Zhang

**Date**

04/03/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

Definition at line 291 of file AuxVector.c.

**9.13 BlaArray.c File Reference**

BLAS1 operations for arrays.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- void `fasp_blas_darray_ax` (const `INT` n, const `REAL` a, `REAL` \*x)
  - $x = a * x$
- void `fasp_blas_darray_axpy` (const `INT` n, const `REAL` a, const `REAL` \*x, `REAL` \*y)
  - $y = a * x + y$
- void `fasp_blas_darray_axpy_nc2` (const `REAL` a, const `REAL` \*x, `REAL` \*y)
  - $y = a * x + y$ , length of x and y should be 2
- void `fasp_blas_darray_axpy_nc3` (const `REAL` a, const `REAL` \*x, `REAL` \*y)
  - $y = a * x + y$ , length of x and y should be 3
- void `fasp_blas_darray_axpy_nc5` (const `REAL` a, const `REAL` \*x, `REAL` \*y)
  - $y = a * x + y$ , length of x and y should be 5
- void `fasp_blas_darray_axpy_nc7` (const `REAL` a, const `REAL` \*x, `REAL` \*y)
  - $y = a * x + y$ , length of x and y should be 7
- void `fasp_blas_darray_axpyz` (const `INT` n, const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)
  - $z = a * x + y$
- void `fasp_blas_darray_axpyz_nc2` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)
  - $z = a * x + y$ , length of x, y and z should be 2
- void `fasp_blas_darray_axpyz_nc3` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)
  - $z = a * x + y$ , length of x, y and z should be 3
- void `fasp_blas_darray_axpyz_nc5` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)
  - $z = a * x + y$ , length of x, y and z should be 5
- void `fasp_blas_darray_axpyz_nc7` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)
  - $z = a * x + y$ , length of x, y and z should be 7
- void `fasp_blas_darray_axpyb` (const `INT` n, const `REAL` a, const `REAL` \*x, const `REAL` b, `REAL` \*y)
  - $y = a * x + b * y$
- `REAL` `fasp_blas_darray_norm1` (const `INT` n, const `REAL` \*x)
  - L1 norm of array x.
- `REAL` `fasp_blas_darray_norm2` (const `INT` n, const `REAL` \*x)
  - L2 norm of array x.
- `REAL` `fasp_blas_darray_norminf` (const `INT` n, const `REAL` \*x)
  - Linf norm of array x.
- `REAL` `fasp_blas_darray_dotprod` (const `INT` n, const `REAL` \*x, const `REAL` \*y)
  - Inner product of two arrays x and y.

### 9.13.1 Detailed Description

BLAS1 operations for arrays.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxThreads.c](#)

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.13.2 Function Documentation

### 9.13.2.1 fasp\_blas\_darray\_ax()

```
void fasp_blas_darray_ax (
    const INT n,
    const REAL a,
    REAL * x )
```

$x = a * x$

#### Parameters

$n$	Number of variables
$a$	Factor a
$x$	Pointer to x

#### Author

Chensong Zhang

#### Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

#### Warning

$x$  is reused to store the resulting array!

Definition at line 43 of file BlaArray.c.

### 9.13.2.2 fasp\_blas\_darray\_axpby()

```
void fasp_blas_darray_axpby (
    const INT n,
    const REAL a,
    const REAL * x,
    const REAL b,
    REAL * y )
```

$y = a * x + b * y$

#### Parameters

$n$	Number of variables
$a$	Factor a
$x$	Pointer to x
$b$	Factor b
$y$	Pointer to y, reused to store the resulting array

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
 Definition at line 580 of file BlaArray.c.

**9.13.2.3 fasp\_blas\_darray\_axpy()**

```
void fasp_blas_darray_axpy (
    const INT n,
    const REAL a,
    const REAL * x,
    REAL * y )
```

 $y = a*x + y$ 
**Parameters**

<i>n</i>	Number of variables
<i>a</i>	Factor a
<i>x</i>	Pointer to x
<i>y</i>	Pointer to y, reused to store the resulting array

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
 Definition at line 93 of file BlaArray.c.

**9.13.2.4 fasp\_blas\_darray\_axpy\_nc2()**

```
void fasp_blas_darray_axpy_nc2 (
    const REAL a,
    const REAL * x,
    REAL * y )
```

 $y = a*x + y$ , length of x and y should be 2
**Parameters**

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array
<i>y</i>	Pointer to the destination array

**Author**

Xiaozhe Hu

**Date**

18/11/2011

Definition at line 170 of file BlaArray.c.

**9.13.2.5 fasp\_blas\_darray\_axpy\_nc3()**

```
void fasp_blas_darray_axpy_nc3 (  
    const REAL a,  
    const REAL * x,  
    REAL * y )
```

y = a\*x + y, length of x and y should be 3

**Parameters**

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array
<i>y</i>	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 193 of file BlaArray.c.

**9.13.2.6 fasp\_blas\_darray\_axpy\_nc5()**

```
void fasp_blas_darray_axpy_nc5 (  
    const REAL a,  
    const REAL * x,  
    REAL * y )
```

y = a\*x + y, length of x and y should be 5

**Parameters**

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array
<i>y</i>	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 222 of file BlaArray.c.

**9.13.2.7 fasp\_blas\_darray\_axpy\_nc7()**

```
void fasp_blas_darray_axpy_nc7 (
    const REAL a,
    const REAL * x,
    REAL * y )
```

$y = a*x + y$ , length of  $x$  and  $y$  should be 7

**Parameters**

$a$	REAL factor $a$
$x$	Pointer to the original array
$y$	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 269 of file BlaArray.c.

**9.13.2.8 fasp\_blas\_darray\_axpyz()**

```
void fasp_blas_darray_axpyz (
    const INT n,
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

$z = a*x + y$

**Parameters**

$n$	Number of variables
$a$	Factor $a$
$x$	Pointer to $x$
$y$	Pointer to $y$
$z$	Pointer to $z$

**Author**

Chensong Zhang



## Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 347 of file BlaArray.c.

**9.13.2.9 fasp\_blas\_darray\_axpyz\_nc2()**

```
void fasp_blas_darray_axpyz_nc2 (
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

$z = a*x + y$ , length of  $x$ ,  $y$  and  $z$  should be 2

## Parameters

$a$	REAL factor $a$
$x$	Pointer to the original array 1
$y$	Pointer to the original array 2
$z$	Pointer to the destination array

## Author

Xiaoze Hu

## Date

18/11/2011

Definition at line 393 of file BlaArray.c.

**9.13.2.10 fasp\_blas\_darray\_axpyz\_nc3()**

```
void fasp_blas_darray_axpyz_nc3 (
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

$z = a*x + y$ , length of  $x$ ,  $y$  and  $z$  should be 3

## Parameters

$a$	REAL factor $a$
$x$	Pointer to the original array 1
$y$	Pointer to the original array 2
$z$	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 419 of file BlaArray.c.

**9.13.2.11 fasp\_blas\_darray\_axpyz\_nc5()**

```
void fasp_blas_darray_axpyz_nc5 (  
    const REAL a,  
    const REAL * x,  
    const REAL * y,  
    REAL * z )
```

$z = a*x + y$ , length of  $x$ ,  $y$  and  $z$  should be 5

**Parameters**

$a$	REAL factor $a$
$x$	Pointer to the original array 1
$y$	Pointer to the original array 2
$z$	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 451 of file BlaArray.c.

**9.13.2.12 fasp\_blas\_darray\_axpyz\_nc7()**

```
void fasp_blas_darray_axpyz_nc7 (  
    const REAL a,  
    const REAL * x,  
    const REAL * y,  
    REAL * z )
```

$z = a*x + y$ , length of  $x$ ,  $y$  and  $z$  should be 7

**Parameters**

$a$	REAL factor $a$
$x$	Pointer to the original array 1
$y$	Pointer to the original array 2
$z$	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 501 of file BlaArray.c.

**9.13.2.13 fasp\_blas\_darray\_dotprod()**

```
REAL fasp_blas_darray_dotprod (  
    const INT n,  
    const REAL * x,  
    const REAL * y )
```

Inner product of two arraies x and y.

**Parameters**

<i>n</i>	Number of variables
<i>x</i>	Pointer to x
<i>y</i>	Pointer to y

**Returns**

Inner product (x,y)

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

Definition at line 741 of file BlaArray.c.

**9.13.2.14 fasp\_blas\_darray\_norm1()**

```
REAL fasp_blas_darray_norm1 (  
    const INT n,  
    const REAL * x )
```

L1 norm of array x.

**Parameters**

<i>n</i>	Number of variables
<i>x</i>	Pointer to x

**Returns**

L1 norm of x

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 628 of file BlaArray.c.

**9.13.2.15 fasp\_blas\_darray\_norm2()**

```
REAL fasp_blas_darray_norm2 (
    const INT n,
    const REAL * x )
```

L2 norm of array x.

**Parameters**

$n$	Number of variables
$x$	Pointer to x

**Returns**

L2 norm of x

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 657 of file BlaArray.c.

**9.13.2.16 fasp\_blas\_darray\_norminf()**

```
REAL fasp_blas_darray_norminf (
    const INT n,
    const REAL * x )
```

Linf norm of array x.

**Parameters**

$n$	Number of variables
$x$	Pointer to x

**Returns**

$L_{\infty}$  norm of  $x$

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Zheng Li on 06/28/2012  
Definition at line 686 of file BlaArray.c.

## 9.14 BlaEigen.c File Reference

Computing the extreme eigenvalues.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- [REAL fasp\\_dcsr\\_maxeig](#) (const [dCSRmat](#) \*A, const [REAL](#) tol, const [INT](#) maxit)  
*Approximate the largest eigenvalue of A by the power method.*

### 9.14.1 Detailed Description

Computing the extreme eigenvalues.

**Note**

This file contains Level-1 (Bla) functions. It requires: [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvCSR.c](#), and [BlaVector.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.14.2 Function Documentation

#### 9.14.2.1 fasp\_dcsr\_maxeig()

```
REAL fasp_dcsr_maxeig (
    const dCSRmat * A,
    const REAL tol,
    const INT maxit )
```

Approximate the largest eigenvalue of A by the power method.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
<i>tol</i>	Tolerance for stopping the power method
<i>maxit</i>	Max number of iterations

**Returns**

Largest eigenvalue

**Author**

Xiaozhe Hu

**Date**

01/25/2011

Definition at line 37 of file BlaEigen.c.

## 9.15 BlaFormat.c File Reference

Subroutines for matrix format conversion.

```
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
```

**Functions**

- [SHORT fasp\\_format\\_dcoo\\_dcsr](#) (const [dCOOmat](#) \*A, [dCSRmat](#) \*B)  
*Transform a REAL matrix from its IJ format to its CSR format.*
- [SHORT fasp\\_format\\_dcsr\\_dcoo](#) (const [dCSRmat](#) \*A, [dCOOmat](#) \*B)  
*Transform a REAL matrix from its CSR format to its IJ format.*
- [SHORT fasp\\_format\\_dstr\\_dcsr](#) (const [dSTRmat](#) \*A, [dCSRmat](#) \*B)  
*Transfer a 'dSTRmat' type matrix into a 'dCSRmat' type matrix.*
- [dCSRmat fasp\\_format\\_dblc\\_dcsr](#) (const [dBLCmat](#) \*Ab)  
*Form the whole dCSRmat A using blocks given in Ab.*
- [dCSRmat \\* fasp\\_format\\_dcsl\\_dcsr](#) (const [dCSRmat](#) \*A)  
*Convert a dCSRmat into a dCSRLmat.*
- [dCSRmat fasp\\_format\\_dbsr\\_dcsr](#) (const [dBSRmat](#) \*B)  
*Transfer a 'dBSRmat' type matrix into a dCSRmat.*
- [dBSRmat fasp\\_format\\_dcsr\\_dbsr](#) (const [dCSRmat](#) \*A, const [INT](#) nb)  
*Transfer a dCSRmat type matrix into a dBSRmat.*
- [dBSRmat fasp\\_format\\_dstr\\_dbsr](#) (const [dSTRmat](#) \*B)  
*Transfer a 'dSTRmat' type matrix to a 'dBSRmat' type matrix.*
- [dCOOmat \\* fasp\\_format\\_dbsr\\_dcoo](#) (const [dBSRmat](#) \*B)  
*Transfer a 'dBSRmat' type matrix to a 'dCOOmat' type matrix.*

### 9.15.1 Detailed Description

Subroutines for matrix format conversion.

**Note**

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), and [BlaSparseCSRL.c](#)

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.15.2 Function Documentation

### 9.15.2.1 fasp\_format\_dblc\_dcsr()

```
dCSRmat fasp_format_dblc_dcsr (
    const dBLCmat * Ab )
```

Form the whole **dCSRmat** A using blocks given in Ab.

#### Parameters

<i>Ab</i>	Pointer to <b>dBLCmat</b> matrix
-----------	----------------------------------

#### Returns

**dCSRmat** matrix if succeed, NULL if fail

#### Author

Shiquan Zhang

#### Date

08/10/2010

Definition at line 294 of file BlaFormat.c.

### 9.15.2.2 fasp\_format\_dbsr\_dcoo()

```
dCOOmat * fasp_format_dbsr_dcoo (
    const dBSRmat * B )
```

Transfer a '**dBSRmat**' type matrix to a '**dCOOmat**' type matrix.

#### Parameters

<i>B</i>	Pointer to <b>dBSRmat</b> matrix
----------	----------------------------------

#### Returns

Pointer to **dCOOmat** matrix

#### Author

Zhiyang Zhou

#### Date

2010/10/26

Definition at line 948 of file BlaFormat.c.

**9.15.2.3 fasp\_format\_dbsr\_dcsr()**

```
dCSRmat fasp_format_dbsr_dcsr (
    const dBSRmat * B )
```

Transfer a 'dBSRmat' type matrix into a dCSRmat.

**Parameters**

<i>B</i>	Pointer to dBSRmat matrix
----------	---------------------------

**Returns**

dCSRmat matrix

**Author**

Zhiyang Zhou

**Date**

10/23/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/24/2012

**Note**

Works for general nb (Xiaozhe)

Definition at line 497 of file BlaFormat.c.

**9.15.2.4 fasp\_format\_dcoo\_dcsr()**

```
SHORT fasp_format_dcoo_dcsr (
    const dCOOmat * A,
    dCSRmat * B )
```

Transform a REAL matrix from its IJ format to its CSR format.

**Parameters**

<i>A</i>	Pointer to dCOOmat matrix
<i>B</i>	Pointer to dCSRmat matrix

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Xuehai Huang

**Date**

08/10/2009

Definition at line 36 of file BlaFormat.c.



**9.15.2.5 fasp\_format\_dcsr\_dbsr()**

```
dBSRmat fasp_format_dcsr_dbsr (
    const dCSRmat * A,
    const INT nb )
```

Transfer a [dCSRmat](#) type matrix into a [dBSRmat](#).

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> type matrix
<i>nb</i>	size of each block

**Returns**

[dBSRmat](#) matrix

**Author**

Zheng Li

**Date**

03/27/2014

**Note**

modified by Xiaozhe Hu to avoid potential memory leakage problem

Definition at line 723 of file BlaFormat.c.

**9.15.2.6 fasp\_format\_dcsr\_dcoo()**

```
SHORT fasp_format_dcsr_dcoo (
    const dCSRmat * A,
    dCOOmat * B )
```

Transform a REAL matrix from its CSR format to its IJ format.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix
<i>B</i>	Pointer to <a href="#">dCOOmat</a> matrix

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Xuehai Huang

**Date**

08/10/2009

Modified by Chunsheng Feng, Zheng Li on 10/12/2012

Definition at line 83 of file BlaFormat.c.

### 9.15.2.7 fasp\_format\_dcsr\_dcsr()

```
dCSRmat * fasp_format_dcsr_dcsr (
    const dCSRmat * A )
```

Convert a `dCSRmat` into a `dCSRmat`.

#### Parameters

<i>A</i>	Pointer to <code>dCSRmat</code> matrix
----------	--

#### Returns

Pointer to `dCSRmat` matrix

#### Author

Zhiyang Zhou

#### Date

2011/01/07

Definition at line 363 of file `BlaFormat.c`.

### 9.15.2.8 fasp\_format\_dstr\_dbsr()

```
dBSRmat fasp_format_dstr_dbsr (
    const dSTRmat * B )
```

Transfer a '`dSTRmat`' type matrix to a '`dBSRmat`' type matrix.

#### Parameters

<i>B</i>	Pointer to <code>dSTRmat</code> matrix
----------	--

#### Returns

`dBSRmat` matrix

#### Author

Zhiyang Zhou

#### Date

2010/10/26

Definition at line 844 of file `BlaFormat.c`.

### 9.15.2.9 fasp\_format\_dstr\_dcsr()

```
SHORT fasp_format_dstr_dcsr (
    const dSTRmat * A,
    dCSRmat * B )
```

Transfer a '`dSTRmat`' type matrix into a '`dCSRmat`' type matrix.

## Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> matrix
<i>B</i>	Pointer to <a href="#">dCSRmat</a> matrix

## Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

## Author

Zhiyang Zhou

## Date

2010/04/29

Definition at line 119 of file BlaFormat.c.

## 9.16 BlalLU.c File Reference

Incomplete LU decomposition: ILUK, ILUt, ILUtp.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_iluk](#) (INT n, REAL \*a, INT \*ja, INT \*ia, INT lfil, REAL \*alu, INT \*jlu, INT iwk, INT \*ierr, INT \*nzlu)  
*Get ILU factorization with level of fill-in  $k$  (ilu(k)) for a CSR matrix A.*
- void [fasp\\_ilut](#) (INT n, REAL \*a, INT \*ja, INT \*ia, INT lfil, REAL droptol, REAL \*alu, INT \*jlu, INT iwk, INT \*ierr, INT \*nz)  
*Get incomplete LU factorization with dual truncations of a CSR matrix A.*
- void [fasp\\_ilutp](#) (INT n, REAL \*a, INT \*ja, INT \*ia, INT lfil, REAL droptol, REAL permtol, INT mbloc, REAL \*alu, INT \*jlu, INT \*iperm, INT iwk, INT \*ierr, INT \*nz)  
*Get incomplete LU factorization with pivoting dual truncations of a CSR matrix A.*
- void [fasp\\_symbfactor](#) (INT n, INT \*colind, INT \*rwptr, INT levfill, INT nzmax, INT \*nzlu, INT \*ijlu, INT \*uptr, INT \*ierr)  
*Symbolic factorization of a CSR matrix A in compressed sparse row format, with resulting factors stored in a single MSR data structure.*

### 9.16.1 Detailed Description

Incomplete LU decomposition: ILUK, ILUt, ILUtp.

## Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#)

Translated from SparseKit (Fortran code) by Chunsheng Feng, 09/03/2016  
Copyright (C) 2016–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.16.2 Function Documentation

### 9.16.2.1 fasp\_iluk()

```
void fasp_iluk (
    INT n,
    REAL * a,
    INT * ja,
    INT * ia,
    INT lfil,
    REAL * alu,
    INT * jlu,
    INT iwk,
    INT * ierr,
    INT * nzlu )
```

Get ILU factorization with level of fill-in k (ilu(k)) for a CSR matrix A.

#### Parameters

<i>n</i>	row number of A
<i>a</i>	nonzero entries of A
<i>ja</i>	integer array of column for A
<i>ia</i>	integer array of row pointers for A
<i>lfil</i>	integer. The fill-in parameter. Each row of L and each row of U will have a maximum of lfil elements (excluding the diagonal element). lfil must be .ge. 0.
<i>alu</i>	matrix stored in Modified Sparse Row (MSR) format containing the L and U factors together. The diagonal (stored in alu(1:n) ) is inverted. Each i-th row of the alu,jlu matrix contains the i-th row of L (excluding the diagonal entry=1) followed by the i-th row of U.
<i>jlu</i>	integer array of length n containing the pointers to the beginning of each row of U in the matrix alu,jlu.
<i>iwk</i>	integer. The minimum length of arrays alu, jlu, and levs.
<i>ierr</i>	integer pointer. Return error message with the following meaning. 0 --> successful return. >0 --> zero pivot encountered at step number ierr. -1 --> Error. input matrix may be wrong. (The elimination process has generated a row in L or U whose length is .gt. n.) -2 --> The matrix L overflows the array al. -3 --> The matrix U overflows the array alu. -4 --> Illegal value for lfil. -5 --> zero row encountered.
<i>nzlu</i>	integer pointer. Return number of nonzero entries for alu and jlu

#### Note

: All the diagonal elements of the input matrix must be nonzero.

#### Author

Chunsheng Feng

#### Date

09/06/2016

Definition at line 72 of file BlalLU.c.

### 9.16.2.2 fasp\_ilut()

```
void fasp_ilut (
    INT n,
    REAL * a,
    INT * ja,
    INT * ia,
    INT lfil,
    REAL droptol,
    REAL * alu,
    INT * jlu,
    INT iwk,
    INT * ierr,
    INT * nz )
```

Get incomplete LU factorization with dual truncations of a CSR matrix A.

#### Parameters

<i>n</i>	row number of A
<i>a</i>	nonzero entries of A
<i>ja</i>	integer array of column for A
<i>ia</i>	integer array of row pointers for A
<i>lfil</i>	integer. The fill-in parameter. Each row of L and each row of U will have a maximum of lfil elements (excluding the diagonal element). lfil must be .ge. 0.
<i>droptol</i>	real*8. Sets the threshold for dropping small terms in the factorization. See below for details on dropping strategy.
<i>alu</i>	matrix stored in Modified Sparse Row (MSR) format containing the L and U factors together. The diagonal (stored in alu(1:n) ) is inverted. Each i-th row of the alu,jlu matrix contains the i-th row of L (excluding the diagonal entry=1) followed by the i-th row of U.
<i>jlu</i>	integer array of length n containing the pointers to the beginning of each row of U in the matrix alu,jlu.
<i>iwk</i>	integer. The lengths of arrays alu and jlu. If the arrays are not big enough to store the ILU factorizations, ilut will stop with an error message.
<i>ierr</i>	integer pointer. Return error message with the following meaning. 0 --> successful return. >0 --> zero pivot encountered at step number ierr. -1 --> Error. input matrix may be wrong. (The elimination process has generated a row in L or U whose length is .gt. n.) -2 --> The matrix L overflows the array al. -3 --> The matrix U overflows the array alu. -4 --> Illegal value for lfil. -5 --> zero row encountered.
<i>nz</i>	integer pointer. Return number of nonzero entries for alu and jlu

#### Note

All the diagonal elements of the input matrix must be nonzero.

#### Author

Chunsheng Feng

#### Date

09/06/2016

Definition at line 467 of file BlalLU.c.

### 9.16.2.3 fasp\_ilutp()

```
void fasp_ilutp (
    INT n,
    REAL * a,
    INT * ja,
    INT * ia,
    INT lfil,
    REAL droptol,
    REAL permtol,
    INT mbloc,
    REAL * alu,
    INT * jlu,
    INT * iperm,
    INT iwk,
    INT * ierr,
    INT * nz )
```

Get incomplete LU factorization with pivoting dual truncations of a CSR matrix A.

#### Parameters

<i>n</i>	row number of A
<i>a</i>	nonzero entries of A
<i>ja</i>	integer array of column for A
<i>ia</i>	integer array of row pointers for A
<i>lfil</i>	integer. The fill-in parameter. Each row of L and each row of U will have a maximum of lfil elements (excluding the diagonal element). lfil must be .ge. 0.
<i>droptol</i>	real*8. Sets the threshold for dropping small terms in the factorization. See below for details on dropping strategy.
<i>permtol</i>	tolerance ratio used to determine whether or not to permute two columns. At step i columns i and j are permuted when $\text{abs}(a(i,j)) * \text{permtol} > \text{abs}(a(i,i))$ [0 --> never permute; good values 0.1 to 0.01]
<i>mbloc</i>	integer. If desired, permuting can be done only within the diagonal blocks of size mbloc. Useful for PDE problems with several degrees of freedom.. If feature not wanted take mbloc=n.
<i>alu</i>	matrix stored in Modified Sparse Row (MSR) format containing the L and U factors together. The diagonal (stored in alu(1:n) ) is inverted. Each i-th row of the alu,jlu matrix contains the i-th row of L (excluding the diagonal entry=1) followed by the i-th row of U.
<i>jlu</i>	integer array of length n containing the pointers to the beginning of each row of U in the matrix alu,jlu.
<i>iperm</i>	permutation arrays
<i>iwk</i>	integer. The lengths of arrays alu and jlu. If the arrays are not big enough to store the ILU factorizations, ilut will stop with an error message.
<i>ierr</i>	integer pointer. Return error message with the following meaning. 0 --> successful return. >0 --> zero pivot encountered at step number ierr. -1 --> Error. input matrix may be wrong. (The elimination process has generated a row in L or U whose length is .gt. n.) -2 --> The matrix L overflows the array al. -3 --> The matrix U overflows the array alu. -4 --> Illegal value for lfil. -5 --> zero row encountered.
<i>nz</i>	integer pointer. Return number of nonzero entries for alu and jlu

#### Note

: All the diagonal elements of the input matrix must be nonzero.

**Author**

Chunsheng Feng

**Date**

09/06/2016

Definition at line 906 of file BlalLU.c.

**9.16.2.4 fasp\_symbfactor()**

```
void fasp_symbfactor (
    INT n,
    INT * colind,
    INT * rwptr,
    INT levfill,
    INT nzmax,
    INT * nzlu,
    INT * ijlu,
    INT * uptr,
    INT * ierr )
```

Symbolic factorization of a CSR matrix A in compressed sparse row format, with resulting factors stored in a single MSR data structure.

**Parameters**

<i>n</i>	row number of A
<i>colind</i>	integer array of column for A
<i>rwptr</i>	integer array of row pointers for A
<i>levfill</i>	integer. Level of fill-in allowed
<i>nzmax</i>	integer. The maximum number of nonzero entries in the approximate factorization of a. This is the amount of storage allocated for ijlu.
<i>nzlu</i>	integer pointer. Return number of nonzero entries for alu and jlu
<i>ijlu</i>	integer array of length nzlu containing pointers to delimit rows and specify column number for stored elements of the approximate factors of A. the L and U factors are stored as one matrix.
<i>uptr</i>	integer array of length n containing the pointers to upper trig matrix
<i>ierr</i>	integer pointer. Return error message with the following meaning. 0 --> successful return. 1 --> not enough storage; check mneed.

**Author**

Chunsheng Feng

**Date**

09/06/2016

Symbolic factorization of a matrix in compressed sparse row format, \* with resulting factors stored in a single MSR data structure. \*

This routine uses the CSR data structure of A in two integer vectors \* colind, rwptr to set up the data structure for the ILU(levfill) \* factorization of A in the integer vectors ijlu and uptr. Both L \* and U are stored in the same structure, and uptr(i) is the pointer \* to the beginning of the i-th row of U in ijlu. \*

Method Used \* ===== \*

The implementation assumes that the diagonal entries are \* nonzero, and remain nonzero throughout the elimination \* process. The algorithm proceeds row by row. When computing \* the sparsity pattern of the i-th row, the effect of row \* operations from previous rows is considered. Only those \* preceding rows j for which (i,j) is nonzero need be considered, \* since otherwise we would not have formed a linear combination \* of rows i and j. \*

The method used has some variations possible. The definition \* of ILU(s) is not well specified enough to get a factorization \* that is uniquely defined, even in the sparsity pattern that \* results. For s = 0 or 1, there is not much variation, but for \* higher levels of fill the problem is as follows: Suppose \* during the decomposition while computing the nonzero pattern \* for row i the following principal submatrix is obtained: \*

Furthermore, suppose that entry (i,j) resulted from an earlier \* fill-in and has level s1, and (j,k) resulted from an earlier \* fill-in and has level s2: \*

When using A(j,j) to annihilate A(i,j), fill-in will be incurred \* in A(i,k). How should its level be defined? It would not be \* operated on if A(i,j) or A(j,m) had not been filled in. The \* version used here is to define its level as s1 + s2 + 1. However, \* other reasonable choices would have been min(s1,s2) or max(s1,s2). \* Using the sum gives a more conservative strategy in terms of the \* growth of the number of nonzeros as s increases. \*

levels(n+2:nzlu) stores the levels from previous rows, \* that is, the s2's above. levels(1:n) stores the fill-levels \* of the current row (row i), which are the s1's above. \* levels(n+1) is not used, so levels is conformant with MSR format. \*

Vectors used: \* ===== \*

lastcol(n): \* The integer lastcol(k) is the row index of the last row \* to have a nonzero in column k, including the current \* row, and fill-in up to this point. So for the matrix \*

```
-----
* | 11 12 15 | * | 21 22 26 | * | 32 33 34 | * | 41 43 44 | * | 52 54 55 56 | * | 62 66 | * -----
----- *
```

after step 1, lastcol() = [1 0 0 0 1 0] \* after step 2, lastcol() = [2 2 0 0 2 2] \* after step 3, lastcol() = [2 3 3 3 2 3] \* after step 4, lastcol() = [4 3 4 4 4 3] \* after step 5, lastcol() = [4 5 4 5 5 5] \* after step 6, lastcol() = [4 6 4 5 5 6] \*

Note that on step 2, lastcol(5) = 2 because there is a \* fillin position (2,5) in the matrix. lastcol() is used \* to determine if a nonzero occurs in column j because \* it is a nonzero in the original matrix, or was a fill. \*

rowll(n): \* The integer vector rowll is used to keep a linked list of \* the nonzeros in the current row, allowing fill-in to be \* introduced sensibly. rowll is initialized with the \* original nonzeros of the current row, and then sorted \* using a shell sort. A pointer called head \* (what ingenuity) is initialized. Note that at any \* point rowll may contain garbage left over from previous \* rows, which the linked list structure skips over. \* For row 4 of the matrix above, first rowll is set to \* rowll() = [3 1 2 5 -], where - indicates any integer. \* Then the vector is sorted, which yields \* rowll() = [1 2 3 5 -]. The vector is then expanded \* to linked list form by setting head = 1 and \* rowll() = [2 3 5 - 7 -], where 7 indicates termination. \*

ijlu(nzlu): \* The returned nonzero structure for the LU factors. \* This is built up row by row in MSR format, with both L \* and U stored in the data structure. Another vector, uptr(n), \* is used to give pointers to the beginning of the upper \* triangular part of the LU factors in ijlu. \*

levels(n+2:nzlu): \* This vector stores the fill level for each entry from \* all the previous rows, used to compute if the current entry \* will exceed the allowed levels of fill. The value in \* levels(m) is added to the level of fill for the element in \* the current row that is being reduced, to figure if \* a column entry is to be accepted as fill, or rejected. \* See the method explanation above. \*

levels(1:n): \* This vector stores the fill level number for the current \* row's entries. If they were created as fill elements \* themselves, this number is added to the corresponding \* entry in levels(n+2:nzlu) to see if a particular column \* entry will \* be created as new fill or not. NOTE: in practice, the \* value in levels(1:n) is one larger than the "fill" level of \* the corresponding row entry, except for the diagonal \* entry. That is why the accept/reject test in the code \* is "if (levels(j) + levels(m) .le. levfill + 1)". \*

## on entry:

n = The order of the matrix A. ija = Integer array. Matrix A stored in modified sparse row format. levfill = Integer. Level of fill-in allowed. nzmax = Integer. The maximum number of nonzero entries in the approximate factorization of a. This is the amount of storage allocated for ijlu.



**on return:**

nzlu = The actual number of entries in the approximate factors, plus one. ijl = Integer array of length nzlu containing pointers to delimit rows and specify column number for stored elements of the approximate factors of a. the l and u factors are stored as one matrix. uptr = Integer array of length n containing the pointers to upper trig matrix  
ierr is an error flag: ierr = -i --> near zero pivot in step i ierr = 0 --> all's OK ierr = 1 --> not enough storage; check mneed. ierr = 2 --> illegal parameter  
mneed = contains the actual number of elements in ldu, or the amount of additional storage needed for ldu

**work arrays:**

lastcol = integer array of length n containing last update of the corresponding column. levels = integer array of length n containing the level of fill-in in current row in its first n entries, and level of fill of previous rows of U in remaining part.  
rowl = integer array of length n containing pointers to implement a linked list for the fill-in elements.

**external functions:**

ifix, float, min0, srr  
Definition at line 1372 of file BlalLU.c.

---

## 9.17 BlalLUSetupBSR.c File Reference

Setup incomplete LU decomposition for [dBSRmat](#) matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [SHORT fasp\\_ilu\\_dbsr\\_setup](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Get ILU decoposition of a BSR matrix A.*
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_step](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam, [INT](#) step)
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_omp](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Multi-thread ILU decoposition of a BSR matrix A based on graph coloring.*
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_levsch\\_omp](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Get ILU decoposition of a BSR matrix A based on level schedule strategy.*
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_levsch\\_step](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam, [INT](#) step)
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_mc\\_omp](#) ([dBSRmat](#) \*A, [dCSRmat](#) \*Ap, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Multi-thread ILU decoposition of a BSR matrix A based on graph coloring.*

### 9.17.1 Detailed Description

Setup incomplete LU decomposition for [dBSRmat](#) matrices.

**Note**

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxTiming.c](#), [BlaSmallMatInv.c](#), [BlalLU.c](#), [BlaSmallMat.c](#), [BlaSmallMatInv.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), and [PreDataInit.c](#)

Copyright (C) 2010–2020 by the FASP team. All rights reserved.

---

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.17.2 Function Documentation

### 9.17.2.1 fasp\_ilu\_dbsr\_setup()

```
SHORT fasp_ilu_dbsr_setup (
    dBSRmat * A,
    ILU_data * iludata,
    ILU_param * iluparam )
```

Get ILU decoposition of a BSR matrix A.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> matrix
<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
<i>iluparam</i>	Pointer to <a href="#">ILU_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Shiquan Zhang, Xiaozhe Hu

#### Date

11/08/2010

#### Note

Works for general nb (Xiaozhe)

Change the size of work space by Zheng Li 04/26/2015.

Modified by Chunsheng Feng on 08/11/2017 for iludata->type not inited.

Definition at line 54 of file BlalILUSetupBSR.c.

### 9.17.2.2 fasp\_ilu\_dbsr\_setup\_levsch\_omp()

```
SHORT fasp_ilu_dbsr_setup_levsch_omp (
    dBSRmat * A,
    ILU_data * iludata,
    ILU_param * iluparam )
```

Get ILU decoposition of a BSR matrix A based on level schedule strategy.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> matrix
<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
<i>iluparam</i>	Pointer to <a href="#">ILU_param</a>

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Zheng Li

**Date**

12/04/2016

**Note**

Only works for nb = 1, 2, 3 (Zheng)

Modified by Chunsheng Feng on 09/06/2017 for iludata->type not inited

Definition at line 455 of file BlalLUSetupBSR.c.

**9.17.2.3 fasp\_ilu\_dbsr\_setup\_mc\_omp()**

```
SHORT fasp_ilu_dbsr_setup_mc_omp (
    dBSRmat * A,
    dCSRmat * Ap,
    ILU_data * iludata,
    ILU_param * iluparam )
```

Multi-thread ILU decoposition of a BSR matrix A based on graph coloring.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> matrix
<i>Ap</i>	Pointer to <a href="#">dCSRmat</a> matrix which provides sparsity pattern
<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
<i>iluparam</i>	Pointer to <a href="#">ILU_param</a>

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Zheng Li

**Date**

12/04/2016

**Note**

Only works for 1, 2, 3 nb (Zheng)

Modified by Chunsheng Feng on 09/06/2017 for iludata->type not inited.

Definition at line 745 of file BlalLUSetupBSR.c.

### 9.17.2.4 fasp\_ilu\_dbsr\_setup\_omp()

```
SHORT fasp_ilu_dbsr_setup_omp (
    dBSRmat * A,
    ILU_data * iludata,
    ILU_param * iluparam )
```

Multi-thread ILU decoposition of a BSR matrix A based on graph coloring.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> matrix
<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
<i>iluparam</i>	Pointer to <a href="#">ILU_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Zheng Li

#### Date

12/04/2016

#### Note

Only works for 1, 2, 3 nb (Zheng)

Modified by Chunsheng Feng on 09/06/2017 for iludata->type not inited.

Definition at line 319 of file BlalLUSetupBSR.c.

## 9.18 BlalLUSetupCSR.c File Reference

Setup incomplete LU decomposition for [dCSRmat](#) matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [SHORT fasp\\_ilu\\_dcsr\\_setup](#) ([dCSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Get ILU decomposition of a CSR matrix A.*

### 9.18.1 Detailed Description

Setup incomplete LU decomposition for [dCSRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxTiming.c](#), [BlalLU.c](#), [BlaSparseCSR.c](#), and [PreDataInit.c](#)

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.18.2 Function Documentation

### 9.18.2.1 fasp\_ilu\_dcsr\_setup()

```
SHORT fasp_ilu_dcsr_setup (
    dCSRmat * A,
    ILU_data * iludata,
    ILU_param * iluparam )
```

Get ILU decomposition of a CSR matrix A.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix
<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
<i>iluparam</i>	Pointer to <a href="#">ILU_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Shiquan Zhang Xiaozhe Hu

#### Date

12/27/2009

Modified by Chunsheng Feng on 02/12/2017: add iperm array for ILUTp  
Definition at line 40 of file BlalLUSetupCSR.c.

## 9.19 BlalLUSetupSTR.c File Reference

Setup incomplete LU decomposition for [dSTRmat](#) matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_ilu\\_dstr\\_setup0](#) ([dSTRmat](#) \*A, [dSTRmat](#) \*LU)  
*Get ILU(0) decomposition of a structured matrix A.*
- void [fasp\\_ilu\\_dstr\\_setup1](#) ([dSTRmat](#) \*A, [dSTRmat](#) \*LU)  
*Get ILU(1) decomposition of a structured matrix A.*

### 9.19.1 Detailed Description

Setup incomplete LU decomposition for [dSTRmat](#) matrices.

**Note**

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#), [BlaSmallMat.c](#), [BlaSmallMatInv.c](#), [BlaSparseSTR.c](#), and [BlaArray.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

**9.19.2 Function Documentation****9.19.2.1 fasp\_ilu\_dstr\_setup0()**

```
void fasp_ilu_dstr_setup0 (
    dSTRmat * A,
    dSTRmat * LU )
```

Get ILU(0) decomposition of a structured matrix A.

**Parameters**

<i>A</i>	Pointer to <a href="#">dSTRmat</a>
<i>LU</i>	Pointer to ILU structured matrix of REAL type

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

11/08/2010

**Note**

Only works for 5 bands 2D and 7 bands 3D matrix with default offsets (order can be arbitrary)!

Definition at line 38 of file [BlaILUSetupSTR.c](#).

**9.19.2.2 fasp\_ilu\_dstr\_setup1()**

```
void fasp_ilu_dstr_setup1 (
    dSTRmat * A,
    dSTRmat * LU )
```

Get ILU(1) decomposition of a structured matrix A.

**Parameters**

<i>A</i>	Pointer to original structured matrix of REAL type
<i>LU</i>	Pointer to ILU structured matrix of REAL type

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

11/08/2010

**Note**

Put L and U in a STR matrix and it has the following structure: the diag is d, the offdiag of L are alpha1 to alpha6, the offdiag of U are beta1 to beta6

Only works for 5 bands 2D and 7 bands 3D matrix with default offsets

Definition at line 333 of file BlalLUSetupSTR.c.

## 9.20 BlalO.c File Reference

Matrix/vector input/output subroutines.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "hb_io.h"
#include "BlalIOUtil.inl"
```

### Functions

- void [fasp\\_dcsrvec\\_read1](#) (const char \*filename, [dCSRmat](#) \*A, [dvector](#) \*b)  
*Read A and b from a SINGLE disk file.*
- void [fasp\\_dcsrvec\\_read2](#) (const char \*filename, const char \*filerhs, [dCSRmat](#) \*A, [dvector](#) \*b)  
*Read A and b from two separate disk files.*
- void [fasp\\_dcsr\\_read](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in IJ format.*
- void [fasp\\_dcoo\\_read](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in IJ format – indices starting from 0.*
- void [fasp\\_dcoo\\_read1](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in IJ format – indices starting from 1.*
- void [fasp\\_dcoo\\_shift\\_read](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in IJ format – indices starting from 0.*
- void [fasp\\_dmtx\\_read](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in MatrixMarket general format.*
- void [fasp\\_dmtxsym\\_read](#) (const char \*filename, [dCSRmat](#) \*A)  
*Read A from matrix disk file in MatrixMarket sym format.*
- void [fasp\\_dstr\\_read](#) (const char \*filename, [dSTRmat](#) \*A)  
*Read A from a disk file in dSTRmat format.*
- void [fasp\\_dbsr\\_read](#) (const char \*filename, [dBSRmat](#) \*A)  
*Read A from a disk file in dBSRmat format.*
- void [fasp\\_dvecind\\_read](#) (const char \*filename, [dvector](#) \*b)
- void [fasp\\_dvec\\_read](#) (const char \*filename, [dvector](#) \*b)  
*Read b from a disk file in array format.*
- void [fasp\\_ivecind\\_read](#) (const char \*filename, [ivector](#) \*b)  
*Read b from matrix disk file.*
- void [fasp\\_ivec\\_read](#) (const char \*filename, [ivector](#) \*b)  
*Read b from a disk file in array format.*

- void [fasp\\_dcsrvec\\_write1](#) (const char \*filename, [dCSRmat](#) \*A, [dvector](#) \*b)  
*Write A and b to a SINGLE disk file.*
- void [fasp\\_dcsrvec\\_write2](#) (const char \*filename, const char \*filerhs, [dCSRmat](#) \*A, [dvector](#) \*b)  
*Write A and b to two separate disk files.*
- void [fasp\\_dcoo\\_write](#) (const char \*filename, [dCSRmat](#) \*A)  
*Write a matrix to disk file in IJ format (coordinate format)*
- void [fasp\\_dstr\\_write](#) (const char \*filename, [dSTRmat](#) \*A)  
*Write a dSTRmat to a disk file.*
- void [fasp\\_dbsr\\_print](#) (const char \*filename, [dBSRmat](#) \*A)  
*Print a dBSRmat to a disk file in a readable format.*
- void [fasp\\_dbsr\\_write](#) (const char \*filename, [dBSRmat](#) \*A)  
*Write a dBSRmat to a disk file.*
- void [fasp\\_dvec\\_write](#) (const char \*filename, [dvector](#) \*vec)  
*Write a dvector to disk file.*
- void [fasp\\_dvecind\\_write](#) (const char \*filename, [dvector](#) \*vec)  
*Write a dvector to disk file in coordinate format.*
- void [fasp\\_ivec\\_write](#) (const char \*filename, [ivector](#) \*vec)  
*Write a ivector to disk file in coordinate format.*
- void [fasp\\_dvec\\_print](#) (const [INT](#) n, [dvector](#) \*u)  
*Print first n entries of a vector of REAL type.*
- void [fasp\\_ivec\\_print](#) (const [INT](#) n, [ivector](#) \*u)  
*Print first n entries of a vector of INT type.*
- void [fasp\\_dcsr\\_print](#) (const [dCSRmat](#) \*A)  
*Print out a dCSRmat matrix in coordinate format.*
- void [fasp\\_dcoo\\_print](#) (const [dCOOmat](#) \*A)  
*Print out a dCOOmat matrix in coordinate format.*
- void [fasp\\_dbsr\\_write\\_coo](#) (const char \*filename, const [dBSRmat](#) \*A)  
*Print out a dBSRmat matrix in coordinate format for matlab spy.*
- void [fasp\\_dcsr\\_write\\_coo](#) (const char \*filename, const [dCSRmat](#) \*A)  
*Print out a dCSRmat matrix in coordinate format for matlab spy.*
- void [fasp\\_dstr\\_print](#) (const [dSTRmat](#) \*A)  
*Print out a dSTRmat matrix in coordinate format.*
- void [fasp\\_matrix\\_read](#) (const char \*filename, void \*A)  
*Read matrix from different kinds of formats from both ASCII and binary files.*
- void [fasp\\_matrix\\_read\\_bin](#) (const char \*filename, void \*A)  
*Read matrix in binary format.*
- void [fasp\\_matrix\\_write](#) (const char \*filename, void \*A, const [INT](#) flag)  
*write matrix from different kinds of formats from both ASCII and binary files*
- void [fasp\\_vector\\_read](#) (const char \*filerhs, void \*b)  
*Read RHS vector from different kinds of formats in ASCII or binary files.*
- void [fasp\\_vector\\_write](#) (const char \*filerhs, void \*b, const [INT](#) flag)  
*write RHS vector from different kinds of formats in both ASCII and binary files*
- void [fasp\\_hb\\_read](#) (const char \*input\_file, [dCSRmat](#) \*A, [dvector](#) \*b)  
*Read matrix and right-hans side from a HB format file.*



## Variables

- int [ilength](#)
- int [dlength](#)

### 9.20.1 Detailed Description

Matrix/vector input/output subroutines.

#### Note

Read, write or print a matrix or a vector in various formats

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxConvert.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaFormat.c](#), [BlaSparseBSR.c](#), [BlaSparseCOO.c](#), [BlaSparseCSR.c](#), and [BlaSpmvCSR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.20.2 Function Documentation

#### 9.20.2.1 fasp\_dbsr\_print()

```
void fasp_dbsr_print (
    const char * filename,
    dBSRmat * A )
```

Print a [dBSRmat](#) to a disk file in a readable format.

#### Parameters

<i>filename</i>	File name for A
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix A

#### Author

Chensong Zhang

#### Date

01/07/2021

Definition at line 1221 of file BlaiO.c.

#### 9.20.2.2 fasp\_dbsr\_read()

```
void fasp_dbsr_read (
    const char * filename,
    dBSRmat * A )
```

Read A from a disk file in [dBSRmat](#) format.

#### Parameters

<i>filename</i>	File name for matrix A
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> A

**Note**

This routine reads a [dBSRmat](#) matrix from a disk file in the following format:

File format:

- ROW, COL, NNZ
- nb: size of each block
- storage\_manner: storage manner of each block
- ROW+1: length of IA
- IA(i), i=0:ROW
- NNZ: length of JA
- JA(i), i=0:NNZ-1
- NNZ\*nb\*nb: length of val
- val(i), i=0:NNZ\*nb\*nb-1

**Author**

Xiaozhe Hu

**Date**

10/29/2010

Definition at line 728 of file BlaiO.c.

**9.20.2.3 fasp\_dbsr\_write()**

```
void fasp_dbsr_write (
    const char * filename,
    dBSRmat * A )
```

Write a [dBSRmat](#) to a disk file.

**Parameters**

<i>filename</i>	File name for A
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix A

**Note**

The routine writes the specified REAL vector in BSR format. Refer to the reading subroutine [fasp\\_dbsr\\_read](#).

**Author**

Shiquan Zhang

**Date**

10/29/2010

Definition at line 1266 of file BlaiO.c.

#### 9.20.2.4 fasp\_dbsr\_write\_coo()

```
void fasp_dbsr_write_coo (
    const char * filename,
    const dBSRmat * A )
```

Print out a **dBSRmat** matrix in coordinate format for matlab spy.

##### Parameters

<i>filename</i>	Name of file to write to
<i>A</i>	Pointer to the <b>dBSRmat</b> matrix A

##### Author

Chunsheng Feng

##### Date

11/14/2013

Modified by Chensong Zhang on 06/14/2014: Fix index problem.  
Definition at line 1504 of file BlaiO.c.

#### 9.20.2.5 fasp\_dcoo\_print()

```
void fasp_dcoo_print (
    const dCOOmat * A )
```

Print out a **dCOOmat** matrix in coordinate format.

##### Parameters

<i>A</i>	Pointer to the <b>dCOOmat</b> matrix A
----------	--

##### Author

Ziteng Wang

##### Date

12/24/2012

Definition at line 1481 of file BlaiO.c.

#### 9.20.2.6 fasp\_dcoo\_read()

```
void fasp_dcoo_read (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in IJ format – indices starting from 0.

##### Parameters

<i>filename</i>	File name for matrix
<i>A</i>	Pointer to the CSR matrix

**Note**

File format:

- nrow ncol nnz % number of rows, number of columns, and nnz
- i j a\_ij % i, j a\_ij in each line

After reading, it converts the matrix to [dCSRmat](#) format.

**Author**

Xuehai Huang, Chensong Zhang

**Date**

03/29/2009

Definition at line 326 of file BlalO.c.

**9.20.2.7 fasp\_dcoo\_read1()**

```
void fasp_dcoo_read1 (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in IJ format – indices starting from 1.

**Parameters**

<i>filename</i>	File name for matrix
<i>A</i>	Pointer to the CSR matrix

**Note**

File format:

- nrow ncol nnz % number of rows, number of columns, and nnz
- i j a\_ij % i, j a\_ij in each line

**Author**

Xiaozhe Hu, Chensong Zhang

**Date**

03/24/2013

Modified by Chensong Zhang on 01/12/2019 : Convert COO to CSR

Definition at line 378 of file BlalO.c.

**9.20.2.8 fasp\_dcoo\_shift\_read()**

```
void fasp_dcoo_shift_read (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in IJ format – indices starting from 0.

## Parameters

<i>filename</i>	File name for matrix
<i>A</i>	Pointer to the CSR matrix

## Note

File format:

- nrow ncol nnz % number of rows, number of columns, and nnz
- i j a\_ij % i, j a\_ij in each line

i and j suppose to start with index 1!!!

After read in, it shifts the index to C fashion and converts the matrix to [dCSRmat](#) format.

## Author

Xiaozhe Hu

## Date

04/01/2014

Definition at line 433 of file BlaiO.c.

## 9.20.2.9 fasp\_dcoo\_write()

```
void fasp_dcoo_write (
    const char * filename,
    dCSRmat * A )
```

Write a matrix to disk file in IJ format (coordinate format)

## Parameters

<i>A</i>	pointer to the <a href="#">dCSRmat</a> matrix
<i>filename</i>	char for vector file name

## Note

The routine writes the specified REAL vector in COO format. Refer to the reading subroutine [fasp\\_dcoo\\_read](#).

File format:

- The first line of the file gives the number of rows, the number of columns, and the number of nonzeros.
- Then gives nonzero values in i j a(i,j) format.

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 1134 of file BlaiO.c.

**9.20.2.10 fasp\_dcsr\_print()**

```
void fasp_dcsr_print (
    const dCSRmat * A )
```

Print out a **dCSRmat** matrix in coordinate format.

**Parameters**

<i>A</i>	Pointer to the <b>dCSRmat</b> matrix A
----------	--

**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 1459 of file BlalO.c.

**9.20.2.11 fasp\_dcsr\_read()**

```
void fasp_dcsr_read (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in IJ format.

**Parameters**

<i>filename</i>	Char for matrix file name
<i>A</i>	Pointer to the CSR matrix

**Author**

Ziteng Wang

**Date**

12/25/2012

Definition at line 251 of file BlalO.c.

**9.20.2.12 fasp\_dcsr\_write\_coo()**

```
void fasp_dcsr_write_coo (
    const char * filename,
    const dCSRmat * A )
```

Print out a **dCSRmat** matrix in coordinate format for matlab spy.

**Parameters**

<i>filename</i>	Name of file to write to
<i>A</i>	Pointer to the <b>dCSRmat</b> matrix A

**Author**

Chunsheng Feng

**Date**

11/14/2013

Definition at line 1558 of file BlalO.c.

**9.20.2.13 fasp\_dcsrvec\_read1()**

```
void fasp_dcsrvec_read1 (
    const char * filename,
    dCSRmat * A,
    dvector * b )
```

Read A and b from a SINGLE disk file.

**Parameters**

<i>filename</i>	File name
<i>A</i>	Pointer to the CSR matrix
<i>b</i>	Pointer to the dvector

**Note**

This routine reads a [dCSRmat](#) matrix and a dvector vector from a single disk file. The difference between this and `fasp_dcoovec_read` is that this routine support non-square matrices.

File format:

- nrow ncol % number of rows and number of columns
- ia(j), j=0:nrow % row index
- ja(j), j=0:nnz-1 % column index
- a(j), j=0:nnz-1 % entry value
- n % number of entries
- b(j), j=0:n-1 % entry value

**Author**

Xuehai Huang

**Date**

03/29/2009

Modified by Chensong Zhang on 03/14/2012

Definition at line 63 of file BlalO.c.

**9.20.2.14 fasp\_dcsrvec\_read2()**

```
void fasp_dcsrvec_read2 (
    const char * filemat,
```

```
const char * filerhs,  
dCSRmat * A,  
dvector * b )
```

Read *A* and *b* from two separate disk files.



## Parameters

<i>filemat</i>	File name for matrix
<i>filerhs</i>	File name for right-hand side
<i>A</i>	Pointer to the dCSR matrix
<i>b</i>	Pointer to the dvector

## Note

This routine reads a [dCSRmat](#) matrix and a dvector vector from a disk file.

CSR matrix file format:

- *nrow* % number of columns (rows)
- *ia*(*j*), *j*=0:*nrow* % row index
- *ja*(*j*), *j*=0:*nnz*-1 % column index
- *a*(*j*), *j*=0:*nnz*-1 % entry value

RHS file format:

- *n* % number of entries
- *b*(*j*), *j*=0:*nrow*-1 % entry value

Indices start from 1, NOT 0!!!

## Author

Zhiyang Zhou

## Date

2010/08/06

Modified by Chensong Zhang on 2012/01/05  
Definition at line 162 of file BlalO.c.

## 9.20.2.15 fasp\_dcsrvec\_write1()

```
void fasp_dcsrvec_writel (
    const char * filename,
    dCSRmat * A,
    dvector * b )
```

Write A and b to a SINGLE disk file.

## Parameters

<i>filename</i>	File name
<i>A</i>	Pointer to the CSR matrix
<i>b</i>	Pointer to the dvector

**Note**

This routine writes a [dCSRmat](#) matrix and a dvector vector to a single disk file.

File format:

- nrow ncol % number of rows and number of columns
- ia(j), j=0:nrow % row index
- ja(j), j=0:nnz-1 % column index
- a(j), j=0:nnz-1 % entry value
- n % number of entries
- b(j), j=0:n-1 % entry value

**Author**

Feiteng Huang

**Date**

05/19/2012

Modified by Chensong on 12/26/2012

Definition at line 1002 of file BlalO.c.

**9.20.2.16 fasp\_dcsrvec\_write2()**

```
void fasp_dcsrvec_write2 (
    const char * filemat,
    const char * filerhs,
    dCSRmat * A,
    dvector * b )
```

Write A and b to two separate disk files.

**Parameters**

<i>filemat</i>	File name for matrix
<i>filerhs</i>	File name for right-hand side
<i>A</i>	Pointer to the dCSR matrix
<i>b</i>	Pointer to the dvector

**Note**

This routine writes a [dCSRmat](#) matrix and a dvector vector to two disk files.

CSR matrix file format:

- nrow % number of columns (rows)
- ia(j), j=0:nrow % row index
- ja(j), j=0:nnz-1 % column index
- a(j), j=0:nnz-1 % entry value

RHS file format:

- n % number of entries

- `b(j)`, `j=0:nrow-1` % entry value

Indices start from 1, NOT 0!!!

**Author**

Feiteng Huang

**Date**

05/19/2012

Definition at line 1070 of file BlalO.c.

**9.20.2.17 fasp\_dmtx\_read()**

```
void fasp_dmtx_read (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in MatrixMarket general format.

**Parameters**

<i>filename</i>	File name for matrix
<i>A</i>	Pointer to the CSR matrix

**Note**

File format: This routine reads a MatrixMarket general matrix from a mtx file. And it converts the matrix to [dCSRmat](http://math.nist.gov/MatrixMarket/) format. For details of mtx format, please refer to <http://math.nist.gov/MatrixMarket/>.

Indices start from 1, NOT 0!!!

**Author**

Chensong Zhang

**Date**

09/05/2011

Definition at line 486 of file BlalO.c.

**9.20.2.18 fasp\_dmtxsym\_read()**

```
void fasp_dmtxsym_read (
    const char * filename,
    dCSRmat * A )
```

Read A from matrix disk file in MatrixMarket sym format.

**Parameters**

<i>filename</i>	File name for matrix
<i>A</i>	Pointer to the CSR matrix

**Note**

File format: This routine reads a MatrixMarket symmetric matrix from a mtx file. And it converts the matrix to [dCSRmat](#) format. For details of mtx format, please refer to <http://math.nist.gov/MatrixMarket/>.  
Indices start from 1, NOT 0!!!

**Author**

Chensong Zhang

**Date**

09/02/2011

Definition at line 545 of file BlalO.c.

**9.20.2.19 fasp\_dstr\_print()**

```
void fasp_dstr_print (
    const dSTRmat * A )
```

Print out a [dSTRmat](#) matrix in coordinate format.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix A
----------	---

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1597 of file BlalO.c.

**9.20.2.20 fasp\_dstr\_read()**

```
void fasp_dstr_read (
    const char * filename,
    dSTRmat * A )
```

Read A from a disk file in [dSTRmat](#) format.

**Parameters**

<i>filename</i>	File name for the matrix
<i>A</i>	Pointer to the <a href="#">dSTRmat</a>

**Note**

This routine reads a [dSTRmat](#) matrix from a disk file. After done, it converts the matrix to [dCSRmat](#) format.  
File format:

- nx, ny, nz

- nc: number of components
- nband: number of bands
- n: size of diagonal, you must have diagonal
- diag(j), j=0:n-1
- offset, length: offset and length of off-diag1
- offdiag(j), j=0:length-1

**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 622 of file BlaiO.c.

**9.20.2.21 fasp\_dstr\_write()**

```
void fasp_dstr_write (
    const char * filename,
    dSTRmat * A )
```

Write a [dSTRmat](#) to a disk file.**Parameters**

<i>filename</i>	File name for A
<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix A

**Note**The routine writes the specified REAL vector in STR format. Refer to the reading subroutine [fasp\\_dstr\\_read](#).**Author**

Shiquan Zhang

**Date**

03/29/2010

Definition at line 1169 of file BlaiO.c.

**9.20.2.22 fasp\_dvec\_print()**

```
void fasp_dvec_print (
    const INT n,
    dvector * u )
```

Print first n entries of a vector of REAL type.

## Parameters

<i>n</i>	An interger (if n=0, then print all entries)
<i>u</i>	Pointer to a dvector

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 1416 of file BlalO.c.

**9.20.2.23 fasp\_dvec\_read()**

```
void fasp_dvec_read (
    const char * filename,
    dvector * b )
```

Read b from a disk file in array format.

## Parameters

<i>filename</i>	File name for vector b
<i>b</i>	Pointer to the dvector b (output)

## Note

File Format:

- nrow
- val\_j, j=0:nrow-1

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 858 of file BlalO.c.

**9.20.2.24 fasp\_dvec\_write()**

```
void fasp_dvec_write (
    const char * filename,
    dvector * vec )
```

Write a dvector to disk file.

## Parameters

<i>vec</i>	Pointer to the dvector
<i>filename</i>	File name

## Author

Xuehai Huang

## Date

03/29/2009

Definition at line 1319 of file BlaiO.c.

**9.20.2.25 fasp\_dvecind\_write()**

```
void fasp_dvecind_write (
    const char * filename,
    dvector * vec )
```

Write a dvector to disk file in coordinate format.

## Parameters

<i>vec</i>	Pointer to the dvector
<i>filename</i>	File name

## Note

The routine writes the specified REAL vector in IJ format.

- The first line of the file is the length of the vector;
- After that, each line gives index and value of the entries.

## Author

Xuehai Huang

## Date

03/29/2009

Definition at line 1352 of file BlaiO.c.

**9.20.2.26 fasp\_hb\_read()**

```
fasp_hb_read (
    const char * input_file,
    dCSRmat * A,
    dvector * b )
```

Read matrix and right-hans side from a HB format file.

**Parameters**

<i>input_file</i>	File name of vector file
<i>A</i>	Pointer to the matrix
<i>b</i>	Pointer to the vector

**Note**

Modified from the C code hb\_io\_prb.c by John Burkardt, which is NOT part of the FASP project!

**Author**

Xiaoehe Hu

**Date**

05/30/2014

Definition at line 2102 of file BlalO.c.

**9.20.2.27 fasp\_ivec\_print()**

```
void fasp_ivec_print (
    const INT n,
    ivector * u )
```

Print first n entries of a vector of INT type.

**Parameters**

<i>n</i>	An interger (if n=0, then print all entries)
<i>u</i>	Pointer to an ivector

**Author**

Chensong Zhang

**Date**

03/29/2009

Definition at line 1438 of file BlalO.c.

**9.20.2.28 fasp\_ivec\_read()**

```
void fasp_ivec_read (
    const char * filename,
    ivector * b )
```

Read b from a disk file in array format.

**Parameters**

<i>filename</i>	File name for vector b
<i>b</i>	Pointer to the dvector b (output)



**Note**

File Format:

- nrow
- val<sub>j</sub>, j=0:nrow-1

**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 951 of file BlaiO.c.

**9.20.2.29 fasp\_ivec\_write()**

```
void fasp_ivec_write (
    const char * filename,
    ivector * vec )
```

Write a ivector to disk file in coordinate format.

**Parameters**

<i>vec</i>	Pointer to the dvector
<i>filename</i>	File name

**Note**

The routine writes the specified INT vector in IJ format.

- The first line of the file is the length of the vector;
- After that, each line gives index and value of the entries.

**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 1385 of file BlaiO.c.

**9.20.2.30 fasp\_ivecind\_read()**

```
void fasp_ivecind_read (
    const char * filename,
    ivector * b )
```

Read b from matrix disk file.

**Parameters**

<i>filename</i>	File name for vector b
<i>b</i>	Pointer to the dvector b (output)

**Note**

File Format:

- nrow
- ind\_j, val\_j ... j=0:nrow-1

**Author**

Chensong Zhang

**Date**

03/29/2009

Definition at line 910 of file BlalO.c.

**9.20.2.31 fasp\_matrix\_read()**

```
fasp_matrix_read (
    const char * filename,
    void * A )
```

Read matrix from different kinds of formats from both ASCII and binary files.

**Parameters**

<i>filename</i>	File name of matrix file
<i>A</i>	Pointer to the matrix

**Note**

Flags for matrix file format:

- fileflag % fileflag = 1: binary, fileflag = 0000: ASCII
- formatflag % a 3-digit number for internal use, see below
- matrix % different types of matrix

Meaning of formatflag:

- matrixflag % first digit of formatflag
  - matrixflag = 1: CSR format
  - matrixflag = 2: BSR format
  - matrixflag = 3: STR format
  - matrixflag = 4: COO format
  - matrixflag = 5: MTX format
  - matrixflag = 6: MTX symmetrical format
- ilength % third digit of formatflag, length of INT

- `length % fourth digit of formatflag`, length of REAL

**Author**

Ziteng Wang

**Date**

12/24/2012

Modified by Chensong Zhang on 05/01/2013  
 Definition at line 1631 of file BlalO.c.

**9.20.2.32 fasp\_matrix\_read\_bin()**

```
void fasp_matrix_read_bin (
    const char * filename,
    void * A )
```

Read matrix in binary format.

**Parameters**

<i>filename</i>	File name of matrix file
<i>A</i>	Pointer to the matrix

**Author**

Xiaozhe Hu

**Date**

04/14/2013

Modified by Chensong Zhang on 05/01/2013: Use it to read binary files!!!  
 Definition at line 1740 of file BlalO.c.

**9.20.2.33 fasp\_matrix\_write()**

```
fasp_matrix_write (
    const char * filename,
    void * A,
    const INT flag )
```

write matrix from different kinds of formats from both ASCII and binary files

**Parameters**

<i>filename</i>	File name of matrix file
<i>A</i>	Pointer to the matrix
<i>flag</i>	Type of file and matrix, a 3-digit number

**Note**

Meaning of flag:

- fileflag % fileflag = 1: binary, fileflag = 0: ASCII
- matrixflag
  - matrixflag = 1: CSR format
  - matrixflag = 2: BSR format
  - matrixflag = 3: STR format

Matrix file format:

- fileflag % fileflag = 1: binary, fileflag = 0000: ASCII
- formatflag % a 3-digit number
- matrixflag % different kinds of matrix judged by formatflag

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1813 of file BlaiO.c.

**9.20.2.34 fasp\_vector\_read()**

```
fasp_vector_read (
    const char * filerhs,
    void * b )
```

Read RHS vector from different kinds of formats in ASCII or binary files.

**Parameters**

<i>filerhs</i>	File name of vector file
<i>b</i>	Pointer to the vector

**Note**

Matrix file format:

- fileflag % fileflag = 1: binary, fileflag = 0000: ASCII
- formatflag % a 3-digit number
- vector % different kinds of vector judged by formatflag

Meaning of formatflag:

- vectorflag % first digit of formatflag
  - vectorflag = 1: dvec format
  - vectorflag = 2: ivec format
  - vectorflag = 3: dvecind format
  - vectorflag = 4: ivecind format

- ilength % second digit of formatflag, length of INT
- dlength % third digit of formatflag, length of REAL

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1905 of file BlaiO.c.

**9.20.2.35 fasp\_vector\_write()**

```
fasp_vector_write (
    const char * filerhs,
    void * b,
    const INT flag )
```

write RHS vector from different kinds of formats in both ASCII and binary files

**Parameters**

<i>filerhs</i>	File name of vector file
<i>b</i>	Pointer to the vector
<i>flag</i>	Type of file and vector, a 2-digit number

**Note**

Meaning of the flags

- fileflag % fileflag = 1: binary, fileflag = 0: ASCII
- vectorflag
  - vectorflag = 1: dvec format
  - vectorflag = 2: ivec format
  - vectorflag = 3: dvecind format
  - vectorflag = 4: ivecind format

Matrix file format:

- fileflag % fileflag = 1: binary, fileflag = 0000: ASCII
- formatflag % a 2-digit number
- vectorflag % different kinds of vector judged by formatflag

**Author**

Ziteng Wang

**Date**

12/24/2012

Modified by Chensong Zhang on 05/02/2013: fix a bug when writing in binary format  
Definition at line 2013 of file BlaiO.c.

### 9.20.3 Variable Documentation

#### 9.20.3.1 dlength

```
int dlength
Length of REAL in byte
Definition at line 24 of file BlaIO.c.
```

#### 9.20.3.2 ilength

```
int ilength
Length of INT in byte
Definition at line 23 of file BlaIO.c.
```

## 9.21 BlaOrderingCSR.c File Reference

Generating ordering using algebraic information.  
`#include "fasp.h"`

### Functions

- void `fasp_dcsr_CMK_order` (const `dCSRmat` \*A, `INT` \*order, `INT` \*oindex)  
*Ordering vertices of matrix graph corresponding to A.*
- void `fasp_dcsr_RCMK_order` (const `dCSRmat` \*A, `INT` \*order, `INT` \*oindex, `INT` \*rorder)  
*Reverse CMK ordering.*

#### 9.21.1 Detailed Description

Generating ordering using algebraic information.

##### Note

This file contains Level-1 (Bla) functions.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.21.2 Function Documentation

##### 9.21.2.1 fasp\_dcsr\_CMK\_order()

```
void fasp_dcsr_CMK_order (
    const dCSRmat * A,
    INT * order,
    INT * oindex )
```

Ordering vertices of matrix graph corresponding to A.

## Parameters

<i>A</i>	Pointer to matrix
<i>oindex</i>	Pointer to index of vertices in order
<i>order</i>	Pointer to vertices with increasing degree

## Author

Zheng Li, Chensong Zhang

## Date

05/28/2014

Definition at line 37 of file BlaOrderingCSR.c.

**9.21.2.2 fasp\_dcsr\_RCMK\_order()**

```
void fasp_dcsr_RCMK_order (
    const dCSRmat * A,
    INT * order,
    INT * oindex,
    INT * rorder )
```

Reverse CMK ordering.

## Parameters

<i>A</i>	Pointer to matrix
<i>order</i>	Pointer to vertices with increasing degree
<i>oindex</i>	Pointer to index of vertices in order
<i>rorder</i>	Pointer to reverse order

## Author

Zheng Li, Chensong Zhang

## Date

10/10/2014

Definition at line 87 of file BlaOrderingCSR.c.

**9.22 BlaSchwarzSetup.c File Reference**

Setup phase for the Schwarz methods.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- `INT fasp_swz_dcsr_setup (SWZ_data *swzdata, SWZ_param *swzparam)`  
*Setup phase for the Schwarz methods.*
- `void fasp_dcsr_swz_forward (SWZ_data *swzdata, SWZ_param *swzparam, dvector *x, dvector *b)`  
*Schwarz smoother: forward sweep.*
- `void fasp_dcsr_swz_backward (SWZ_data *swzdata, SWZ_param *swzparam, dvector *x, dvector *b)`  
*Schwarz smoother: backward sweep.*

### 9.22.1 Detailed Description

Setup phase for the Schwarz methods.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#), [AuxVector.c](#), [BlaSparseCSR.c](#), [BlaSparseUtil.c](#), and [KryPvgmres.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.22.2 Function Documentation

#### 9.22.2.1 fasp\_dcsr\_swz\_backward()

```
void fasp_dcsr_swz_backward (
    SWZ_data * swzdata,
    SWZ_param * swzparam,
    dvector * x,
    dvector * b )
```

Schwarz smoother: backward sweep.

#### Parameters

<i>swzdata</i>	Pointer to the Schwarz data
<i>swzparam</i>	Pointer to the Schwarz parameter
<i>x</i>	Pointer to solution vector
<i>b</i>	Pointer to right hand

#### Author

Zheng Li, Chensong Zhang

#### Date

2014/10/5

Definition at line 325 of file BlaSchwarzSetup.c.

#### 9.22.2.2 fasp\_dcsr\_swz\_forward()

```
void fasp_dcsr_swz_forward (
    SWZ_data * swzdata,
```



```

    SWZ_param * swzparam,
    dvector * x,
    dvector * b )

```

Schwarz smoother: forward sweep.

#### Parameters

<i>swzdata</i>	Pointer to the Schwarz data
<i>swzparam</i>	Pointer to the Schwarz parameter
<i>x</i>	Pointer to solution vector
<i>b</i>	Pointer to right hand

#### Author

Zheng Li, Chensong Zhang

#### Date

2014/10/5

Definition at line 216 of file BlaSchwarzSetup.c.

#### 9.22.2.3 fasp\_swz\_dcsr\_setup()

```

INT fasp_swz_dcsr_setup (
    SWZ_data * swzdata,
    SWZ_param * swzparam )

```

Setup phase for the Schwarz methods.

#### Parameters

<i>swzdata</i>	Pointer to the Schwarz data
<i>swzparam</i>	Type of the Schwarz method

#### Returns

FASP\_SUCCESS if succeed

#### Author

Ludmil, Xiaozhe Hu

#### Date

03/22/2011

Modified by Zheng Li on 10/09/2014

Definition at line 47 of file BlaSchwarzSetup.c.

## 9.23 BlaSmallMat.c File Reference

BLAS operations for *small* dense matrices.

```

#include "fasp.h"
#include "fasp_functs.h"

```

## Functions

- void `fasp_blas_smat_axm` (`REAL *a`, `const INT n`, `const REAL alpha`)  
*Compute  $a = \alpha * a$  (in place)*
- void `fasp_blas_smat_add` (`const REAL *a`, `const REAL *b`, `const INT n`, `const REAL alpha`, `const REAL beta`, `REAL *c`)  
*Compute  $c = \alpha * a + \beta * b$ .*
- void `fasp_blas_smat_m xv_nc2` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the product of a 2\*2 matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m xv_nc3` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the product of a 3\*3 matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m xv_nc4` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the product of a 4\*4 matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m xv_nc5` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the product of a 5\*5 matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m xv_nc7` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the product of a 7\*7 matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m xv` (`const REAL *a`, `const REAL *b`, `REAL *c`, `const INT n`)  
*Compute the product of a small full matrix a and a array b, stored in c.*
- void `fasp_blas_smat_m ul_nc2` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the matrix product of two 2\* matrices a and b, stored in c.*
- void `fasp_blas_smat_m ul_nc3` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the matrix product of two 3\*3 matrices a and b, stored in c.*
- void `fasp_blas_smat_m ul_nc4` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the matrix product of two 4\*4 matrices a and b, stored in c.*
- void `fasp_blas_smat_m ul_nc5` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the matrix product of two 5\*5 matrices a and b, stored in c.*
- void `fasp_blas_smat_m ul_nc7` (`const REAL *a`, `const REAL *b`, `REAL *c`)  
*Compute the matrix product of two 7\*7 matrices a and b, stored in c.*
- void `fasp_blas_smat_m ul` (`const REAL *a`, `const REAL *b`, `REAL *c`, `const INT n`)  
*Compute the matrix product of two small full matrices a and b, stored in c.*
- void `fasp_blas_smat_y pAx_nc2` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y + Ax$ , where 'A' is a 2\*2 dense matrix.*
- void `fasp_blas_smat_y pAx_nc3` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y + Ax$ , where 'A' is a 3\*3 dense matrix.*
- void `fasp_blas_smat_y pAx_nc4` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y + Ax$ , where 'A' is a 4\*4 dense matrix.*
- void `fasp_blas_smat_y pAx_nc5` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y + Ax$ , where 'A' is a 5\*5 dense matrix.*
- void `fasp_blas_smat_y pAx_nc7` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y + Ax$ , where 'A' is a 7\*7 dense matrix.*
- void `fasp_blas_smat_y pAx` (`const REAL *A`, `const REAL *x`, `REAL *y`, `const INT n`)  
*Compute  $y := y + Ax$ , where 'A' is a n\*n dense matrix.*
- void `fasp_blas_smat_y mA x_nc2` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y - Ax$ , where 'A' is a 2\*2 dense matrix.*
- void `fasp_blas_smat_y mA x_nc3` (`const REAL *A`, `const REAL *x`, `REAL *y`)  
*Compute  $y := y - Ax$ , where 'A' is a 3\*3 dense matrix.*
- void `fasp_blas_smat_y mA x_nc4` (`const REAL *A`, `const REAL *x`, `REAL *y`)

Compute  $y := y - Ax$ , where 'A' is a 4\*4 dense matrix.

- void `fasp_blas_smat_ymAx_nc5` (const `REAL` \*A, const `REAL` \*x, `REAL` \*y)

Compute  $y := y - Ax$ , where 'A' is a 5\*5 dense matrix.

- void `fasp_blas_smat_ymAx_nc7` (const `REAL` \*A, const `REAL` \*x, `REAL` \*y)

Compute  $y := y - Ax$ , where 'A' is a 7\*7 dense matrix.

- void `fasp_blas_smat_ymAx` (const `REAL` \*A, const `REAL` \*x, `REAL` \*y, const `INT` n)

Compute  $y := y - Ax$ , where 'A' is a n\*n dense matrix.

- void `fasp_blas_smat_aAxpby` (const `REAL` alpha, const `REAL` \*A, const `REAL` \*x, const `REAL` beta, `REAL` \*y, const `INT` n)

Compute  $y := \alpha A x + \beta y$

### 9.23.1 Detailed Description

BLAS operations for *small* dense matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), and [PreDataInit.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

#### Warning

These routines are designed for full matrices only!

This file contains very long lines. Not print friendly!

### 9.23.2 Function Documentation

#### 9.23.2.1 `fasp_blas_smat_aAxpby()`

```
void fasp_blas_smat_aAxpby (
    const REAL alpha,
    const REAL * A,
    const REAL * x,
    const REAL beta,
    REAL * y,
    const INT n )
```

Compute  $y := \alpha A x + \beta y$

#### Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the REAL array which stands for a n*n full matrix
<i>x</i>	Pointer to the REAL array with length n
<i>beta</i>	REAL factor beta
<i>y</i>	Pointer to the REAL array with length n
<i>n</i>	Length of array x and y

**Author**

Zhiyang Zhou, Chensong Zhang

**Date**

2010/10/25

Definition at line 1064 of file BlaSmallMat.c.

**9.23.2.2 fasp\_blas\_smat\_add()**

```
void fasp_blas_smat_add (
    const REAL * a,
    const REAL * b,
    const INT n,
    const REAL alpha,
    const REAL beta,
    REAL * c )
```

Compute  $c = \alpha * a + \beta * b$ .**Parameters**

<i>a</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>b</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>n</i>	Dimension of the matrix
<i>alpha</i>	Scalar
<i>beta</i>	Scalar
<i>c</i>	Pointer to the REAL array which stands a $n \times n$ matrix

**Author**

Xiaozhe Hu, Chensong Zhang

**Date**

05/26/2014

Definition at line 65 of file BlaSmallMat.c.

**9.23.2.3 fasp\_blas\_smat\_axm()**

```
void fasp_blas_smat_axm (
    REAL * a,
    const INT n,
    const REAL alpha )
```

Compute  $a = \alpha * a$  (in place)**Parameters**

<i>a</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>n</i>	Dimension of the matrix
<i>alpha</i>	Scalar

**Author**

Xiaozhe Hu, Chensong Zhang

**Date**

05/26/2014

Definition at line 37 of file BlasSmallMat.c.

**9.23.2.4 fasp\_blas\_smat\_mul()**

```
void fasp_blas_smat_mul (
    const REAL * a,
    const REAL * b,
    REAL * c,
    const INT n )
```

Compute the matrix product of two small full matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>b</i>	Pointer to the REAL array which stands a n*n matrix
<i>c</i>	Pointer to the REAL array which stands a n*n matrix
<i>n</i>	Dimension of the matrix

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

04/21/2010

**Author**

Li Zhao, the case of adding n = 4

**Date**

04/18/2021

Definition at line 540 of file BlasSmallMat.c.

**9.23.2.5 fasp\_blas\_smat\_mul\_nc2()**

```
void fasp_blas_smat_mul_nc2 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the matrix product of two 2\* matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>b</i>	Pointer to the REAL array which stands a n*n matrix
<i>c</i>	Pointer to the REAL array which stands a n*n matrix

**Author**

Xiaozhe Hu

**Date**

18/11/2011

Definition at line 275 of file BlaSmallMat.c.

**9.23.2.6 fasp\_blas\_smat\_mul\_nc3()**

```
void fasp_blas_smat_mul_nc3 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the matrix product of two 3\*3 matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>b</i>	Pointer to the REAL array which stands a n*n matrix
<i>c</i>	Pointer to the REAL array which stands a n*n matrix

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 304 of file BlaSmallMat.c.

**9.23.2.7 fasp\_blas\_smat\_mul\_nc4()**

```
void fasp_blas_smat_mul_nc4 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the matrix product of two 4\*4 matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>b</i>	Pointer to the REAL array which stands a n*n matrix
<i>c</i>	Pointer to the REAL array which stands a n*n matrix

**Author**

Li Zhao

**Date**

04/18/2021

Definition at line 341 of file BlasSmallMat.c.

**9.23.2.8 fasp\_blas\_smat\_mul\_nc5()**

```
void fasp_blas_smat_mul_nc5 (  
    const REAL * a,  
    const REAL * b,  
    REAL * c )
```

Compute the matrix product of two 5\*5 matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 5*5 matrix
<i>b</i>	Pointer to the REAL array which stands a 5*5 matrix
<i>c</i>	Pointer to the REAL array which stands a 5*5 matrix

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 388 of file BlasSmallMat.c.

**9.23.2.9 fasp\_blas\_smat\_mul\_nc7()**

```
void fasp_blas_smat_mul_nc7 (  
    const REAL * a,  
    const REAL * b,  
    REAL * c )
```

Compute the matrix product of two 7\*7 matrices a and b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 7*7 matrix
<i>b</i>	Pointer to the REAL array which stands a 7*7 matrix
<i>c</i>	Pointer to the REAL array which stands a 7*7 matrix

**Author**

Xiaozhe Hu, Shiquan Zhang

## Date

05/01/2010

Definition at line 447 of file BlaSmallMat.c.

**9.23.2.10 fasp\_blas\_smat\_m xv()**

```
void fasp_blas_smat_m xv (
    const REAL * a,
    const REAL * b,
    REAL * c,
    const INT n )
```

Compute the product of a small full matrix a and a array b, stored in c.

## Parameters

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>b</i>	Pointer to the REAL array with length n
<i>c</i>	Pointer to the REAL array with length n
<i>n</i>	Dimension of the matrix

## Author

Xiaozhe Hu, Shiquan Zhang

## Date

04/21/2010

## Author

Li Zhao, the case of adding n = 4

## Date

04/18/2021

Definition at line 221 of file BlaSmallMat.c.

**9.23.2.11 fasp\_blas\_smat\_m xv\_nc2()**

```
void fasp_blas_smat_m xv_nc2 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the product of a 2\*2 matrix a and a array b, stored in c.

## Parameters

<i>a</i>	Pointer to the REAL array which stands a 2*2 matrix
<i>b</i>	Pointer to the REAL array with length 2
<i>c</i>	Pointer to the REAL array with length 2



**Author**

Xiaozhe Hu

**Date**

18/11/2010

Definition at line 93 of file BlasSmallMat.c.

**9.23.2.12 fasp\_blas\_smat\_m xv\_nc3()**

```
void fasp_blas_smat_m xv_nc3 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the product of a 3\*3 matrix a and a array b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 3*3 matrix
<i>b</i>	Pointer to the REAL array with length 3
<i>c</i>	Pointer to the REAL array with length 3

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 115 of file BlasSmallMat.c.

**9.23.2.13 fasp\_blas\_smat\_m xv\_nc4()**

```
void fasp_blas_smat_m xv_nc4 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the product of a 4\*4 matrix a and a array b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 4*4 matrix
<i>b</i>	Pointer to the REAL array with length 4
<i>c</i>	Pointer to the REAL array with length 4

**Author**

Li Zhao

**Date**

04/18/2021

Definition at line 138 of file BlaSmallMat.c.

**9.23.2.14 fasp\_blas\_smat\_m xv\_nc5()**

```
void fasp_blas_smat_m xv_nc5 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the product of a 5\*5 matrix a and a array b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 5*5 matrix
<i>b</i>	Pointer to the REAL array with length 5
<i>c</i>	Pointer to the REAL array with length 5

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 162 of file BlaSmallMat.c.

**9.23.2.15 fasp\_blas\_smat\_m xv\_nc7()**

```
void fasp_blas_smat_m xv_nc7 (
    const REAL * a,
    const REAL * b,
    REAL * c )
```

Compute the product of a 7\*7 matrix a and a array b, stored in c.

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 7*7 matrix
<i>b</i>	Pointer to the REAL array with length 7
<i>c</i>	Pointer to the REAL array with length 7

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 188 of file BlaSmallMat.c.

**9.23.2.16 fasp\_blas\_smat\_ymAx()**

```
void fasp_blas_smat_ymAx (
    const REAL * A,
    const REAL * x,
    REAL * y,
    const INT n )
```

Compute  $y := y - Ax$ , where 'A' is a  $n \times n$  dense matrix.

**Parameters**

<i>A</i>	Pointer to the $n \times n$ dense matrix
<i>x</i>	Pointer to the REAL array with length $n$
<i>y</i>	Pointer to the REAL array with length $n$
<i>n</i>	the dimension of the dense matrix

**Author**

Zhiyang Zhou, Xiaozhe Hu, Chensong Zhang

**Date**

2010/10/25

Modified by Chensong Zhang on 01/25/2017

Definition at line 962 of file BlaSmallMat.c.

**9.23.2.17 fasp\_blas\_smat\_ymAx\_nc2()**

```
void fasp_blas_smat_ymAx_nc2 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y - Ax$ , where 'A' is a  $2 \times 2$  dense matrix.

**Parameters**

<i>A</i>	Pointer to the $2 \times 2$ dense matrix
<i>x</i>	Pointer to the REAL array with length 3
<i>y</i>	Pointer to the REAL array with length 3

**Author**

Xiaozhe Hu

**Date**

18/11/2011

**Note**

Works for 2-component

Definition at line 820 of file BlaSmallMat.c.

**9.23.2.18 fasp\_blas\_smat\_ymAx\_nc3()**

```
void fasp_blas_smat_ymAx_nc3 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y - Ax$ , where 'A' is a 3\*3 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 3*3 dense matrix
<i>x</i>	Pointer to the REAL array with length 3
<i>y</i>	Pointer to the REAL array with length 3

**Author**

Xiaozhe Hu, Zhiyang Zhou

**Date**

01/06/2011

**Note**

Works for 3-component

Definition at line 846 of file BlaSmallMat.c.

**9.23.2.19 fasp\_blas\_smat\_ymAx\_nc4()**

```
void fasp_blas_smat_ymAx_nc4 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y - Ax$ , where 'A' is a 4\*4 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 4*4 dense matrix
<i>x</i>	Pointer to the REAL array with length 4
<i>y</i>	Pointer to the REAL array with length 4

**Author**

Li Zhao

**Date**

04/18/2021

**Note**

Works for 4-component

Definition at line 873 of file BlaSmallMat.c.

**9.23.2.20 fasp\_blas\_smat\_ymAx\_nc5()**

```
void fasp_blas_smat_ymAx_nc5 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y - Ax$ , where 'A' is a 5\*5 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 5*5 dense matrix
<i>x</i>	Pointer to the REAL array with length 5
<i>y</i>	Pointer to the REAL array with length 5

**Author**

Xiaozhe Hu, Zhiyang Zhou

**Date**

01/06/2011

**Note**

Works for 5-component

Definition at line 900 of file BlaSmallMat.c.

**9.23.2.21 fasp\_blas\_smat\_ymAx\_nc7()**

```
void fasp_blas_smat_ymAx_nc7 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y - Ax$ , where 'A' is a 7\*7 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 7*7 dense matrix
<i>x</i>	Pointer to the REAL array with length 7
<i>y</i>	Pointer to the REAL array with length 7

**Author**

Xiaozhe Hu, Zhiyang Zhou

**Date**

01/06/2011

**Note**

Works for 7-component

Definition at line 929 of file BlaSmallMat.c.

**9.23.2.22 fasp\_blas\_smat\_ypAx()**

```
void fasp_blas_smat_ypAx (
    const REAL * A,
    const REAL * x,
    REAL * y,
    const INT n )
```

Compute  $y := y + Ax$ , where 'A' is a  $n \times n$  dense matrix.

**Parameters**

<i>A</i>	Pointer to the $n \times n$ dense matrix
<i>x</i>	Pointer to the REAL array with length $n$
<i>y</i>	Pointer to the REAL array with length $n$
<i>n</i>	Dimension of the dense matrix

**Author**

Zhiyang Zhou, Chensong Zhang

**Date**

2010/10/25

Modified by Chensong Zhang on 01/25/2017

Definition at line 720 of file BlaSmallMat.c.

**9.23.2.23 fasp\_blas\_smat\_ypAx\_nc2()**

```
void fasp_blas_smat_ypAx_nc2 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y + Ax$ , where 'A' is a  $2 \times 2$  dense matrix.

**Parameters**

<i>A</i>	Pointer to the $3 \times 3$ dense matrix
<i>x</i>	Pointer to the REAL array with length 3
<i>y</i>	Pointer to the REAL array with length 3

**Author**

Xiaozhe Hu

**Date**

2011/11/18

Definition at line 589 of file BlaSmallMat.c.

**9.23.2.24 fasp\_blas\_smat\_ypAx\_nc3()**

```
void fasp_blas_smat_ypAx_nc3 (
```

```

const REAL * A,
const REAL * x,
REAL * y )

```

Compute  $y := y + Ax$ , where 'A' is a 3\*3 dense matrix.

#### Parameters

<i>A</i>	Pointer to the 3*3 dense matrix
<i>x</i>	Pointer to the REAL array with length 3
<i>y</i>	Pointer to the REAL array with length 3

#### Author

Zhiyang Zhou, Xiaozhe Hu

#### Date

2010/10/25

Definition at line 613 of file BlaSmallMat.c.

#### 9.23.2.25 fasp\_blas\_smat\_ypAx\_nc4()

```

void fasp_blas_smat_ypAx_nc4 (
    const REAL * A,
    const REAL * x,
    REAL * y )

```

Compute  $y := y + Ax$ , where 'A' is a 4\*4 dense matrix.

#### Parameters

<i>A</i>	Pointer to the 4*4 dense matrix
<i>x</i>	Pointer to the REAL array with length 4
<i>y</i>	Pointer to the REAL array with length 4

#### Author

Li Zhao

#### Date

2021/04/18

Definition at line 637 of file BlaSmallMat.c.

#### 9.23.2.26 fasp\_blas\_smat\_ypAx\_nc5()

```

void fasp_blas_smat_ypAx_nc5 (
    const REAL * A,
    const REAL * x,
    REAL * y )

```

Compute  $y := y + Ax$ , where 'A' is a 5\*5 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 5*5 dense matrix
<i>x</i>	Pointer to the REAL array with length 5
<i>y</i>	Pointer to the REAL array with length 5

**Author**

Zhiyang Zhou, Xiaozhe Hu, Chensong Zhang

**Date**

2010/10/25

Definition at line 662 of file BlaSmallMat.c.

**9.23.2.27 fasp\_blas\_smat\_ypAx\_nc7()**

```
void fasp_blas_smat_ypAx_nc7 (
    const REAL * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := y + Ax$ , where 'A' is a 7\*7 dense matrix.

**Parameters**

<i>A</i>	Pointer to the 7*7 dense matrix
<i>x</i>	Pointer to the REAL array with length 7
<i>y</i>	Pointer to the REAL array with length 7

**Author**

Zhiyang Zhou, Xiaozhe Hu, Chensong Zhang

**Date**

2010/10/25

Definition at line 688 of file BlaSmallMat.c.

**9.24 BlaSmallMatInv.c File Reference**

Find inversion of *small* dense matrices in row-major format.

```
#include "fasp.h"
#include "fasp_functs.h"
```

**Macros**

- #define **SWAP**(a, b) {temp=(a);(a)=(b);(b)=temp;}



## Functions

- void [fasp\\_smat\\_inv\\_nc2](#) (REAL \*a)  
*Compute the inverse matrix of a 2\*2 full matrix A (in place)*
- void [fasp\\_smat\\_inv\\_nc3](#) (REAL \*a)  
*Compute the inverse matrix of a 3\*3 full matrix A (in place)*
- void [fasp\\_smat\\_inv\\_nc4](#) (REAL \*a)  
*Compute the inverse matrix of a 4\*4 full matrix A (in place)*
- void [fasp\\_smat\\_inv\\_nc5](#) (REAL \*a)  
*Compute the inverse matrix of a 5\*5 full matrix A (in place)*
- void [fasp\\_smat\\_inv\\_nc7](#) (REAL \*a)  
*Compute the inverse matrix of a 7\*7 matrix a.*
- void [fasp\\_smat\\_inv\\_nc](#) (REAL \*a, const INT n)  
*Compute the inverse of a matrix using Gauss Elimination.*
- [SHORT fasp\\_smat\\_invp\\_nc](#) (REAL \*a, const INT n)  
*Compute the inverse of a matrix using Gauss Elimination with Pivoting.*
- [SHORT fasp\\_smat\\_inv](#) (REAL \*a, const INT n)  
*Compute the inverse matrix of a small full matrix a.*
- [REAL fasp\\_smat\\_Linf](#) (const REAL \*A, const INT n)  
*Compute the L infinity norm of A.*
- void [fasp\\_smat\\_identity\\_nc2](#) (REAL \*a)  
*Set a 2\*2 full matrix to be a identity.*
- void [fasp\\_smat\\_identity\\_nc3](#) (REAL \*a)  
*Set a 3\*3 full matrix to be a identity.*
- void [fasp\\_smat\\_identity\\_nc5](#) (REAL \*a)  
*Set a 5\*5 full matrix to be a identity.*
- void [fasp\\_smat\\_identity\\_nc7](#) (REAL \*a)  
*Set a 7\*7 full matrix to be a identity.*
- void [fasp\\_smat\\_identity](#) (REAL \*a, const INT n, const INT n2)  
*Set a n\*n full matrix to be a identity.*

### 9.24.1 Detailed Description

Find inversion of *small* dense matrices in row-major format.

Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.24.2 Macro Definition Documentation

#### 9.24.2.1 SWAP

```
#define SWAP(  
    a,  
    b ) { temp=(a); (a)=(b); (b)=temp; }
```

swap two numbers

Definition at line 17 of file BlaSmallMatInv.c.

### 9.24.3 Function Documentation

#### 9.24.3.1 fasp\_smat\_identity()

```
void fasp_smat_identity (
    REAL * a,
    const INT n,
    const INT n2 )
```

Set a  $n \times n$  full matrix to be a identity.

##### Parameters

<i>a</i>	Pointer to the REAL vector which stands for a $n \times n$ full matrix
<i>n</i>	Size of full matrix
<i>n2</i>	Length of the REAL vector which stores the $n \times n$ full matrix

##### Author

Xiaozhe Hu

##### Date

2010/12/25

Definition at line 754 of file BlaSmallMatInv.c.

#### 9.24.3.2 fasp\_smat\_identity\_nc2()

```
void fasp_smat_identity_nc2 (
    REAL * a )
```

Set a  $2 \times 2$  full matrix to be a identity.

##### Parameters

<i>a</i>	Pointer to the REAL vector which stands for a $2 \times 2$ full matrix
----------	--

##### Author

Xiaozhe Hu

##### Date

2011/11/18

Definition at line 674 of file BlaSmallMatInv.c.

#### 9.24.3.3 fasp\_smat\_identity\_nc3()

```
void fasp_smat_identity_nc3 (
    REAL * a )
```

Set a  $3 \times 3$  full matrix to be a identity.

**Parameters**

<i>a</i>	Pointer to the REAL vector which stands for a 3*3 full matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

2010/12/25

Definition at line 691 of file BlaSmallMatInv.c.

**9.24.3.4 fasp\_smat\_identity\_nc5()**

```
void fasp_smat_identity_nc5 (
    REAL * a )
```

Set a 5\*5 full matrix to be a identity.

**Parameters**

<i>a</i>	Pointer to the REAL vector which stands for a 5*5 full matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

2010/12/25

Definition at line 708 of file BlaSmallMatInv.c.

**9.24.3.5 fasp\_smat\_identity\_nc7()**

```
void fasp_smat_identity_nc7 (
    REAL * a )
```

Set a 7\*7 full matrix to be a identity.

**Parameters**

<i>a</i>	Pointer to the REAL vector which stands for a 7*7 full matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

2010/12/25

Definition at line 729 of file BlaSmallMatInv.c.

### 9.24.3.6 fasp\_smat\_inv()

```
SHORT fasp_smat_inv (
    REAL * a,
    const INT n )
```

Compute the inverse matrix of a small full matrix a.

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>n</i>	Dimension of the matrix

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

04/21/2010

Definition at line 603 of file BlaSmallMatInv.c.

### 9.24.3.7 fasp\_smat\_inv\_nc()

```
void fasp_smat_inv_nc (
    REAL * a,
    const INT n )
```

Compute the inverse of a matrix using Gauss Elimination.

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>n</i>	Dimension of the matrix

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

05/01/2010

Definition at line 441 of file BlaSmallMatInv.c.

### 9.24.3.8 fasp\_smat\_inv\_nc2()

```
void fasp_smat_inv_nc2 (
    REAL * a )
```

Compute the inverse matrix of a 2\*2 full matrix A (in place)

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a 2*2 matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

18/11/2011

Definition at line 33 of file BlaSmallMatInv.c.

**9.24.3.9 fasp\_smat\_inv\_nc3()**

```
void fasp_smat_inv_nc3 (  
    REAL * a )
```

Compute the inverse matrix of a 3\*3 full matrix A (in place)

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 3*3 matrix
----------	---

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 67 of file BlaSmallMatInv.c.

**9.24.3.10 fasp\_smat\_inv\_nc4()**

```
void fasp_smat_inv_nc4 (  
    REAL * a )
```

Compute the inverse matrix of a 4\*4 full matrix A (in place)

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a 4*4 matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

01/12/2013

Modified by Hongxuan Zhang on 06/13/2014: Fix a bug in M23.  
Definition at line 111 of file BlaSmallMatInv.c.

**9.24.3.11 fasp\_smat\_inv\_nc5()**

```
void fasp_smat_inv_nc5 (  
    REAL * a )
```

Compute the inverse matrix of a 5\*5 full matrix A (in place)

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a 5*5 matrix
----------	---

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

05/01/2010

Definition at line 170 of file BlaSmallMatInv.c.

### 9.24.3.12 fasp\_smat\_inv\_nc7()

```
void fasp_smat_inv_nc7 (
    REAL * a )
```

Compute the inverse matrix of a 7\*7 matrix a.

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a 7*7 matrix
----------	---

#### Note

This is NOT implemented yet!

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

05/01/2010

Definition at line 425 of file BlaSmallMatInv.c.

### 9.24.3.13 fasp\_smat\_invp\_nc()

```
SHORT fasp_smat_invp_nc (
    REAL * a,
    const INT n )
```

Compute the inverse of a matrix using Gauss Elimination with Pivoting.

#### Parameters

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>n</i>	Dimension of the matrix

**Author**

Chensong Zhang

**Date**

04/03/2015

**Note**

This routine is based on gaussj() from "Numerical Recipes in C"!

Definition at line 508 of file BlaSmallMatInv.c.

**9.24.3.14 fasp\_smat\_Linf()**

```

REAL fasp_smat_Linf (
    const REAL * A,
    const INT n )

```

Compute the L infinity norm of A.

**Parameters**

<i>A</i>	Pointer to the n*n dense matrix
<i>n</i>	the dimension of the dense matrix

**Author**

Xiaozhe Hu

**Date**

05/26/2014

Definition at line 646 of file BlaSmallMatInv.c.

**9.25 BlaSmallMatLU.c File Reference**

LU decomposition and direct solver for small dense matrices.

```

#include <math.h>
#include "fasp.h"

```

**Functions**

- [SHORT fasp\\_smat\\_lu\\_decomp](#) (REAL \*A, INT pivot[], const INT n)  
*LU decomposition of A using Doolittle's method.*
- [SHORT fasp\\_smat\\_lu\\_solve](#) (const REAL \*A, REAL b[], const INT pivot[], REAL x[], const INT n)  
*Solving  $Ax=b$  using LU decomposition.*

**9.25.1 Detailed Description**

LU decomposition and direct solver for small dense matrices.

**Note**

This file contains Level-1 (Bla) functions.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

## 9.25.2 Function Documentation

### 9.25.2.1 fasp\_smat\_lu\_decomp()

```
SHORT fasp_smat_lu_decomp (
    REAL * A,
    INT pivot[],
    const INT n )
```

LU decomposition of A using Doolittle's method.

**Parameters**

<i>A</i>	Pointer to the full matrix
<i>pivot</i>	Pivoting positions
<i>n</i>	Size of matrix A

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Note**

Use Doolittle's method to decompose the  $n \times n$  matrix A into a unit lower triangular matrix L and an upper triangular matrix U such that  $A = LU$ . The matrices L and U replace the matrix A. The diagonal elements of L are 1 and are not stored.

The Doolittle method with partial pivoting is: Determine the pivot row and interchange the current row with the pivot row, then assuming that row k is the current row,  $k = 0, \dots, n - 1$  evaluate in order the following pair of expressions  $U[k][j] = A[k][j] - (L[k][0]*U[0][j] + \dots + L[k][k-1]*U[k-1][j])$  for  $j = k, k+1, \dots, n-1$   $L[i][k] = (A[i][k] - (L[i][0]*U[0][k] + \dots + L[i][k-1]*U[k-1][k])) / U[k][k]$  for  $i = k+1, \dots, n-1$ .

**Author**

Xuehai Huang

**Date**

04/02/2009

Definition at line 52 of file BlaSmallMatLU.c.

### 9.25.2.2 fasp\_smat\_lu\_solve()

```
SHORT fasp_smat_lu_solve (
    const REAL * A,
    REAL b[],
```



```
const INT pivot[],
REAL x[],
const INT n )
```

Solving  $Ax=b$  using LU decomposition.

#### Parameters

<i>A</i>	Pointer to the full matrix
<i>b</i>	Right hand side array (b is used as the working array!!!)
<i>pivot</i>	Pivoting positions
<i>x</i>	Pointer to the solution array
<i>n</i>	Size of matrix A

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Note

This routine uses Doolittle's method to solve the linear equation  $Ax = b$ . This routine is called after the matrix A has been decomposed into a product of a unit lower triangular matrix L and an upper triangular matrix U with pivoting. The solution proceeds by solving the linear equation  $Ly = b$  for y and subsequently solving the linear equation  $Ux = y$  for x.

#### Author

Xuehai Huang

#### Date

04/02/2009

Definition at line 124 of file BlaSmallMatLU.c.

## 9.26 BlaSparseBLC.c File Reference

Sparse matrix block operations.

```
#include <time.h>
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_dblc\\_free](#) (dBLCmat \*A)  
*Free block CSR sparse matrix data memory space.*

#### 9.26.1 Detailed Description

Sparse matrix block operations.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#) and [BlaSparseCSR.c](#)

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.26.2 Function Documentation

### 9.26.2.1 fasp\_dblc\_free()

```
void fasp_dblc_free (
    dBLCmat * A )
```

Free block CSR sparse matrix data memory space.

#### Parameters

<b>A</b>	Pointer to the dBLCmat matrix
----------	-------------------------------

#### Author

Xiaozhe Hu

#### Date

04/18/2014

Definition at line 38 of file BlaSparseBLC.c.

## 9.27 BlaSparseBSR.c File Reference

Sparse matrix operations for dBSRmat matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- **dBSRmat fasp\_dbsr\_create** (const **INT** ROW, const **INT** COL, const **INT** NNZ, const **INT** nb, const **INT** storage\_manner)
 

Create BSR sparse matrix data memory space.
- void **fasp\_dbsr\_alloc** (const **INT** ROW, const **INT** COL, const **INT** NNZ, const **INT** nb, const **INT** storage\_manner, **dBSRmat** \*A)
 

Allocate memory space for BSR format sparse matrix.
- void **fasp\_dbsr\_free** (**dBSRmat** \*A)
 

Free memory space for BSR format sparse matrix.
- void **fasp\_dbsr\_cp** (const **dBSRmat** \*A, **dBSRmat** \*B)
 

copy a **dCSRmat** to a new one  $B=A$
- **INT fasp\_dbsr\_trans** (const **dBSRmat** \*A, **dBSRmat** \*AT)
 

Find  $A^T$  from given **dBSRmat** matrix A.
- **SHORT fasp\_dbsr\_getblk** (const **dBSRmat** \*A, const **INT** \*Is, const **INT** \*Js, const **INT** m, const **INT** n, **dBSRmat** \*B)
 

Get a sub BSR matrix of A with specified rows and columns.
- **SHORT fasp\_dbsr\_diagpref** (**dBSRmat** \*A)

Reorder the column and data arrays of a square BSR matrix, so that the first entry in each row is the diagonal one.

- [dvector fasp\\_dbsr\\_getdiaginv](#) (const [dBSRmat](#) \*A)  
Get  $D^{-1}$  of matrix A.
- [dBSRmat fasp\\_dbsr\\_diaginv](#) (const [dBSRmat](#) \*A)  
Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.
- [dBSRmat fasp\\_dbsr\\_diaginv2](#) (const [dBSRmat](#) \*A, [REAL](#) \*diaginv)  
Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.
- [dBSRmat fasp\\_dbsr\\_diaginv3](#) (const [dBSRmat](#) \*A, [REAL](#) \*diaginv)  
Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.
- [dBSRmat fasp\\_dbsr\\_diaginv4](#) (const [dBSRmat](#) \*A, [REAL](#) \*diaginv)  
Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.
- void [fasp\\_dbsr\\_getdiag](#) (INT n, const [dBSRmat](#) \*A, [REAL](#) \*diag)  
Abstract the diagonal blocks of a BSR matrix.
- [dBSRmat fasp\\_dbsr\\_diagLU](#) (const [dBSRmat](#) \*A, [REAL](#) \*DL, [REAL](#) \*DU)  
Compute  $B := DL * A * DU$ . We decompose each diagonal block of A into LDU form and  $DL = \text{diag}(L^{-1})$  and  $DU = \text{diag}(U^{-1})$ .
- [dBSRmat fasp\\_dbsr\\_diagLU2](#) ([dBSRmat](#) \*A, [REAL](#) \*DL, [REAL](#) \*DU)  
Compute  $B := DL * A * DU$ . We decompose each diagonal block of A into LDU form and  $DL = \text{diag}(L^{-1})$  and  $DU = \text{diag}(U^{-1})$ .
- [dBSRmat fasp\\_dbsr\\_perm](#) (const [dBSRmat](#) \*A, const INT \*P)  
Apply permutation of A, i.e.  $A_{\text{perm}} = PAP'$  by the orders given in P.
- INT [fasp\\_dbsr\\_merge\\_col](#) ([dBSRmat](#) \*A)  
Check and merge some same col index in one row.

### 9.27.1 Detailed Description

Sparse matrix operations for [dBSRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaSmallMat.c](#), and [BlaSmallMatInv.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.27.2 Function Documentation

#### 9.27.2.1 fasp\_dbsr\_alloc()

```
void fasp_dbsr_alloc (
    const INT ROW,
    const INT COL,
    const INT NNZ,
    const INT nb,
    const INT storage_manner,
    dBSRmat * A )
```

Allocate memory space for BSR format sparse matrix.

## Parameters

<i>ROW</i>	Number of rows of block
<i>COL</i>	Number of columns of block
<i>NNZ</i>	Number of nonzero blocks
<i>nb</i>	Dimension of each block
<i>storage_manner</i>	Storage manner for each sub-block
<i>A</i>	Pointer to new <a href="#">dBSRmat</a> matrix

## Author

Xiaozhe Hu

## Date

10/26/2010

Definition at line 99 of file BlaSparseBSR.c.

**9.27.2.2 fasp\_dbsr\_cp()**

```
void fasp_dbsr_cp (
    const dBSRmat * A,
    dBSRmat * B )
```

copy a [dCSRmat](#) to a new one B=A

## Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>B</i>	Pointer to the <a href="#">dBSRmat</a> matrix

## Author

Xiaozhe Hu

## Date

08/07/2011

Definition at line 172 of file BlaSparseBSR.c.

**9.27.2.3 fasp\_dbsr\_create()**

```
dBSRmat fasp_dbsr_create (
    const INT ROW,
    const INT COL,
    const INT NNZ,
    const INT nb,
    const INT storage_manner )
```

Create BSR sparse matrix data memory space.

## Parameters

<i>ROW</i>	Number of rows of block
<i>COL</i>	Number of columns of block
<i>NNZ</i>	Number of nonzero blocks
<i>nb</i>	Dimension of each block
<i>storage_manner</i>	Storage manner for each sub-block

## Returns

A The new [dBSRmat](#) matrix

## Author

Xiaozhe Hu

## Date

10/26/2010

Definition at line 45 of file BlaSparseBSR.c.

**9.27.2.4 fasp\_dbsr\_diaginv()**

```
dBSRmat fasp_dbsr_diaginv (
    const dBSRmat * A )
```

Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.

## Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
----------	---

## Author

Zhiyang Zhou

## Date

2010/10/26

## Note

Works for general nb (Xiaozhe)

Modified by Chunsheng Feng, Zheng Li on 08/25/2012 Modified by Chensong Zhang on 09/27/2017

Definition at line 591 of file BlaSparseBSR.c.

**9.27.2.5 fasp\_dbsr\_diaginv2()**

```
dBSRmat fasp_dbsr_diaginv2 (
    const dBSRmat * A,
    REAL * diaginv )
```

Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.

## Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>diaginv</i>	Pointer to the inverses of all the diagonal blocks

## Author

Zhiyang Zhou

## Date

2010/11/07

## Note

Works for general nb (Xiaozhe)

Modified by Chunsheng Feng, Zheng Li on 08/25/2012  
Definition at line 751 of file BlaSparseBSR.c.

**9.27.2.6 fasp\_dbsr\_diaginv3()**

```
dBSRmat fasp_dbsr_diaginv3 (
    const dBSRmat * A,
    REAL * diaginv )
```

Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.

## Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>diaginv</i>	Pointer to the inverses of all the diagonal blocks

## Returns

BSR matrix after diagonal scaling

## Author

Xiaozhe Hu

## Date

12/25/2010

## Note

Works for general nb (Xiaozhe)

Modified by Xiaozhe Hu on 05/26/2012  
Definition at line 857 of file BlaSparseBSR.c.

**9.27.2.7 fasp\_dbsr\_diaginv4()**

```
dBSRmat fasp_dbsr_diaginv4 (
    const dBSRmat * A,
    REAL * diaginv )
```

Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>diaginv</i>	Pointer to the inverses of all the diagonal blocks

**Returns**

BSR matrix after diagonal scaling

**Note**

Works for general nb (Xiaozhe)

A is pre-ordered that the first block of each row is the diagonal block!

**Author**

Xiaozhe Hu

**Date**

03/12/2011

Modified by Chunsheng Feng, Zheng Li on 08/26/2012

Definition at line 1260 of file BlaSparseBSR.c.

**9.27.2.8 fasp\_dbsr\_diagLU()**

```
dBSRmat fasp_dbsr_diagLU (
    const dBSRmat * A,
    REAL * DL,
    REAL * DU )
```

Compute  $B := DL * A * DU$ . We decompose each diagonal block of A into LDU form and  $DL = \text{diag}(L^{-1})$  and  $DU = \text{diag}(U^{-1})$ .

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>DL</i>	Pointer to the $\text{diag}(L^{-1})$
<i>DU</i>	Pointer to the $\text{diag}(U^{-1})$

**Returns**

BSR matrix after scaling

**Author**

Xiaozhe Hu

**Date**

04/02/2014

Definition at line 1593 of file BlaSparseBSR.c.

**9.27.2.9 fasp\_dbsr\_diagLU2()**

```
dBSRmat fasp_dbsr_diagLU2 (
    dBSRmat * A,
    REAL * DL,
    REAL * DU )
```

Compute  $B := DL * A * DU$ . We decompose each diagonal block of  $A$  into LDU form and  $DL = \text{diag}(L^{-1})$  and  $DU = \text{diag}(U^{-1})$ .

**Parameters**

<i>A</i>	Pointer to the <code>dBSRmat</code> matrix
<i>DL</i>	Pointer to the $\text{diag}(L^{-1})$
<i>DU</i>	Pointer to the $\text{diag}(U^{-1})$

**Returns**

BSR matrix after scaling

**Author**

Zheng Li, Xiaozhe Hu

**Date**

06/17/2014

Definition at line 1822 of file BlaSparseBSR.c.

**9.27.2.10 fasp\_dbsr\_diagpref()**

```
SHORT fasp_dbsr_diagpref (
    dBSRmat * A )
```

Reorder the column and data arrays of a square BSR matrix, so that the first entry in each row is the diagonal one.

**Parameters**

<i>A</i>	Pointer to the BSR matrix
----------	---------------------------

**Author**

Xiaozhe Hu



**Date**

03/10/2011

**Author**

Chunsheng Feng, Zheng Li

**Date**

09/02/2012

**Note**

Reordering is done in place.

Definition at line 385 of file BlaSparseBSR.c.

**9.27.2.11 fasp\_dbsr\_free()**

```
void fasp_dbsr_free (
    dBSRmat * A )
```

Free memory space for BSR format sparse matrix.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 146 of file BlaSparseBSR.c.

**9.27.2.12 fasp\_dbsr\_getblk()**

```
SHORT fasp_dbsr_getblk (
    const dBSRmat * A,
    const INT * Is,
    const INT * Js,
    const INT m,
    const INT n,
    dBSRmat * B )
```

Get a sub BSR matrix of A with specified rows and columns.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> BSR matrix
<i>B</i>	Pointer to <a href="#">dBSRmat</a> BSR matrix
<i>Is</i>	Pointer to selected rows

**Parameters**

<i>Js</i>	Pointer to selected columns
<i>m</i>	Number of selected rows
<i>n</i>	Number of selected columns

**Returns**

FASP\_SUCCESS if succeeded, otherwise return error information.

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

12/25/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 287 of file BlaSparseBSR.c.

**9.27.2.13 fasp\_dbsr\_getdiag()**

```
void fasp_dbsr_getdiag (
    INT n,
    const dBSRmat * A,
    REAL * diag )
```

Abstract the diagonal blocks of a BSR matrix.

**Parameters**

<i>n</i>	Number of blocks to get
<i>A</i>	Pointer to the 'dBSRmat' type matrix
<i>diag</i>	Pointer to array which stores the diagonal blocks in row by row manner

**Author**

Zhiyang Zhou

**Date**

2010/10/26

**Note**

Works for general nb (Xiaozhe)

Modified by Chunsheng Feng, Zheng Li on 08/25/2012  
Definition at line 1555 of file BlaSparseBSR.c.

**9.27.2.14 fasp\_dbsr\_getdiaginv()**

```
dvector fasp_dbsr_getdiaginv (
    const dBSRmat * A )
```

Get  $D^{-1}$  of matrix A.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

02/19/2013

**Note**

Works for general nb (Xiaozhe)

Definition at line 486 of file BlaSparseBSR.c.

**9.27.2.15 fasp\_dbsr\_merge\_col()**

```
INT fasp_dbsr_merge_col (
    dBSRmat * A )
```

Check and merge some same col index in one row.

**Parameters**

<i>A</i>	Pointer to the original <a href="#">dBSRmat</a> matrix
----------	--

**Returns**The new merged [dCSRmat](#) matrix**Author**

Chunsheng Feng

**Date**

30/07/2017

Definition at line 2141 of file BlaSparseBSR.c.

**9.27.2.16 fasp\_dbsr\_perm()**

```
dBSRmat fasp_dbsr_perm (
    const dBSRmat * A,
    const INT * P )
```

Apply permutation of A, i.e. Aperm=PAP' by the orders given in P.

**Parameters**

<i>A</i>	Pointer to the original <a href="#">dBSRmat</a> matrix
<i>P</i>	Pointer to the given ordering

**Returns**

The new ordered [dBSRmat](#) matrix if succeed, NULL if fail

**Author**

Zheng Li

**Date**

24/9/2015

**Note**

$P[i] = k$  means k-th row and column become i-th row and column!

Definition at line 2023 of file BlaSparseBSR.c.

**9.27.2.17 fasp\_dbsr\_trans()**

```
INT fasp_dbsr_trans (
    const dBSRmat * A,
    dBSRmat * AT )
```

Find  $A^T$  from given [dBSRmat](#) matrix A.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>AT</i>	Pointer to the transpose of <a href="#">dBSRmat</a> matrix A

**Author**

Chunsheng FENG

**Date**

2011/06/08

Modified by Xiaozhe Hu (08/06/2011)

Definition at line 199 of file BlaSparseBSR.c.

**9.28 BlaSparseCheck.c File Reference**

Check properties of sparse matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- [INT fasp\\_check\\_diagpos](#) (const [dCSRmat](#) \*A)  
*Check positivity of diagonal entries of a CSR sparse matrix.*
- [SHORT fasp\\_check\\_diagzero](#) (const [dCSRmat](#) \*A)

*Check if a CSR sparse matrix has diagonal entries that are very close to zero.*

- `INT fasp_check_diagdom (const dCSRmat *A)`

*Check whether a matrix is diagonally dominant.*

- `INT fasp_check_symm (const dCSRmat *A)`

*Check symmetry of a sparse matrix of CSR format.*

- `void fasp_check_dCSRmat (const dCSRmat *A)`

*Check whether an dCSRmat matrix is supported or not.*

- `SHORT fasp_check_iCSRmat (const iCSRmat *A)`

*Check whether an iCSRmat matrix is valid or not.*

- `void fasp_check_ordering (dCSRmat *A)`

*Check whether each row of A is in ascending order w.r.t. column indices.*

### 9.28.1 Detailed Description

Check properties of sparse matrices.

Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), and [BlaSparseCSR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.28.2 Function Documentation

#### 9.28.2.1 fasp\_check\_dCSRmat()

```
void fasp_check_dCSRmat (
    const dCSRmat * A )
```

Check whether an dCSRmat matrix is supported or not.

Parameters

<code>A</code>	Pointer to the matrix in dCSRmat format
----------------	---

Author

Chensong Zhang

Date

03/29/2009

Definition at line 281 of file BlaSparseCheck.c.

#### 9.28.2.2 fasp\_check\_diagdom()

```
INT fasp_check_diagdom (
    const dCSRmat * A )
```

Check whether a matrix is diagonally dominant.

`INT fasp_check_diagdom (const dCSRmat *A)`

## Parameters

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
----------	---

## Returns

Number of the rows which are not diagonally dominant

## Note

The routine checks whether the sparse matrix is diagonally dominant each row. It will print out the percentage of the rows which are diagonally dominant.

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 114 of file BlasSparseCheck.c.

### 9.28.2.3 fasp\_check\_diagpos()

```
INT fasp_check_diagpos (  
    const dCSRmat * A )
```

Check positivity of diagonal entries of a CSR sparse matrix.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix
----------	---

## Returns

Number of negative diagonal entries

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 35 of file BlasSparseCheck.c.

### 9.28.2.4 fasp\_check\_diagzero()

```
SHORT fasp_check_diagzero (  
    const dCSRmat * A )
```

Check if a CSR sparse matrix has diagonal entries that are very close to zero.

**Parameters**

<i>A</i>	pointer to the <a href="#">dCSRmat</a> matrix
----------	---

**Returns**

FASP\_SUCCESS if no diagonal entry is close to zero, else ERROR

**Author**

Shuo Zhang

**Date**

03/29/2009

Definition at line 72 of file BlaSparseCheck.c.

**9.28.2.5 fasp\_check\_iCSRmat()**

```
SHORT fasp_check_iCSRmat (  
    const iCSRmat * A )
```

Check whether an [iCSRmat](#) matrix is valid or not.

**Parameters**

<i>A</i>	Pointer to the matrix in <a href="#">iCSRmat</a> format
----------	---

**Author**

Shuo Zhang

**Date**

03/29/2009

Definition at line 318 of file BlaSparseCheck.c.

**9.28.2.6 fasp\_check\_ordering()**

```
void fasp_check_ordering (  
    dCSRmat * A )
```

Check whether each row of A is in ascending order w.r.t. column indices.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
----------	---

**Author**

Chensong Zhang



## Date

02/26/2019

Definition at line 357 of file BLASparseCheck.c.

**9.28.2.7 fasp\_check\_symm()**

```

INT fasp_check_symm (
    const dCSRmat * A )

```

Check symmetry of a sparse matrix of CSR format.

## Parameters

<i>A</i>	Pointer to the <i>dCSRmat</i> matrix
----------	--------------------------------------

## Returns

1 and 2 if the structure of the matrix is not symmetric; 0 if the structure of the matrix is symmetric,

## Note

Print the maximal relative difference between matrix and its transpose.

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 159 of file BLASparseCheck.c.

**9.29 BLASparseCOO.c File Reference**Sparse matrix operations for *dCOOmat* matrices.

```

#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"

```

**Functions**

- *dCOOmat* *fasp\_dcoo\_create* (const INT *m*, const INT *n*, const INT *nnz*)  
*Create IJ sparse matrix data memory space.*
- void *fasp\_dcoo\_alloc* (const INT *m*, const INT *n*, const INT *nnz*, *dCOOmat* \**A*)  
*Allocate COO sparse matrix memory space.*
- void *fasp\_dcoo\_free* (*dCOOmat* \**A*)  
*Free IJ sparse matrix data memory space.*
- void *fasp\_dcoo\_shift* (*dCOOmat* \**A*, const INT *offset*)  
*Re-index a REAL matrix in IJ format to make the index starting from 0 or 1.*

### 9.29.1 Detailed Description

Sparse matrix operations for [dCOOmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#) and [AuxThreads.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.29.2 Function Documentation

#### 9.29.2.1 fasp\_dcoo\_alloc()

```
void fasp_dcoo_alloc (
    const INT m,
    const INT n,
    const INT nnz,
    dCOOmat * A )
```

Allocate COO sparse matrix memory space.

#### Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix

#### Author

Xiaozhe Hu

#### Date

03/25/2013

Definition at line 70 of file [BlaSparseCOO.c](#).

#### 9.29.2.2 fasp\_dcoo\_create()

```
dCOOmat fasp_dcoo_create (
    const INT m,
    const INT n,
    const INT nnz )
```

Create IJ sparse matrix data memory space.

#### Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros

**Returns**

A The new dCOOmat matrix

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 42 of file BlasparseCOO.c.

**9.29.2.3 fasp\_dcoo\_free()**

```
void fasp_dcoo_free (
    dCOOmat * A )
```

Free IJ sparse matrix data memory space.

**Parameters**

<i>A</i>	Pointer to the dCOOmat matrix
----------	-------------------------------

**Author**

Chensong Zhang

**Date**

2010/04/03

Definition at line 102 of file BlasparseCOO.c.

**9.29.2.4 fasp\_dcoo\_shift()**

```
void fasp_dcoo_shift (
    dCOOmat * A,
    const INT offset )
```

Re-index a REAL matrix in IJ format to make the index starting from 0 or 1.

**Parameters**

<i>A</i>	Pointer to IJ matrix
<i>offset</i>	Size of offset (1 or -1)

**Author**

Chensong Zhang

Date

2010/04/06

Modified by Chunsheng Feng, Zheng Li on 08/25/2012  
Definition at line 124 of file BlaSparseCOO.c.

## 9.30 BlaSparseCSR.c File Reference

Sparse matrix operations for `dCSRmat` matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- `dCSRmat fasp_dcsr_create` (const `INT` m, const `INT` n, const `INT` nnz)  
*Create CSR sparse matrix data memory space.*
- `iCSRmat fasp_icsr_create` (const `INT` m, const `INT` n, const `INT` nnz)  
*Create CSR sparse matrix data memory space.*
- void `fasp_dcsr_alloc` (const `INT` m, const `INT` n, const `INT` nnz, `dCSRmat` \*A)  
*Allocate CSR sparse matrix memory space.*
- void `fasp_dcsr_free` (`dCSRmat` \*A)  
*Free CSR sparse matrix data memory space.*
- void `fasp_icsr_free` (`iCSRmat` \*A)  
*Free CSR sparse matrix data memory space.*
- `INT fasp_dcsr_bandwidth` (const `dCSRmat` \*A)  
*Get bandwidth of matrix.*
- `dCSRmat fasp_dcsr_perm` (`dCSRmat` \*A, `INT` \*P)  
*Apply permutation of A, i.e.  $A_{perm}=PAP'$  by the orders given in P.*
- void `fasp_dcsr_sort` (`dCSRmat` \*A)  
*Sort each row of A in ascending order w.r.t. column indices.*
- `SHORT fasp_dcsr_getblk` (const `dCSRmat` \*A, const `INT` \*Is, const `INT` \*Js, const `INT` m, const `INT` n, `dCSRmat` \*B)  
*Get a sub CSR matrix of A with specified rows and columns.*
- void `fasp_dcsr_getdiag` (`INT` n, const `dCSRmat` \*A, `dvector` \*diag)  
*Get first n diagonal entries of a CSR matrix A.*
- void `fasp_dcsr_getcol` (const `INT` n, const `dCSRmat` \*A, `REAL` \*col)  
*Get the n-th column of a CSR matrix A.*
- void `fasp_dcsr_diagpref` (`dCSRmat` \*A)  
*Re-order the column and data arrays of a CSR matrix, so that the first entry in each row is the diagonal.*
- `SHORT fasp_dcsr_regdiag` (`dCSRmat` \*A, const `REAL` value)  
*Regularize diagonal entries of a CSR sparse matrix.*
- void `fasp_icsr_cp` (const `iCSRmat` \*A, `iCSRmat` \*B)  
*Copy a iCSRmat to a new one B=A.*
- void `fasp_dcsr_cp` (const `dCSRmat` \*A, `dCSRmat` \*B)  
*copy a dCSRmat to a new one B=A*
- void `fasp_icsr_trans` (const `iCSRmat` \*A, `iCSRmat` \*AT)

Find transpose of *iCSRmat* matrix *A*.

- [INT fasp\\_dcsr\\_trans](#) (const [dCSRmat](#) \*A, [dCSRmat](#) \*AT)

Find transpose of *dCSRmat* matrix *A*.

- void [fasp\\_dcsr\\_transpose](#) ([INT](#) \*row[2], [INT](#) \*col[2], [REAL](#) \*val[2], [INT](#) \*nn, [INT](#) \*tniz)

Transpose of a *dCSRmat* matrix.

- void [fasp\\_dcsr\\_compress](#) (const [dCSRmat](#) \*A, [dCSRmat](#) \*B, const [REAL](#) dtol)

Compress a CSR matrix *A* and store in CSR matrix *B* by dropping small entries  $\text{abs}(a_{ij}) \leq \text{dtol}$ .

- [SHORT fasp\\_dcsr\\_compress\\_inplace](#) ([dCSRmat](#) \*A, const [REAL](#) dtol)

Compress a CSR matrix *A* IN PLACE by dropping small entries  $\text{abs}(a_{ij}) \leq \text{dtol}$ .

- void [fasp\\_dcsr\\_shift](#) ([dCSRmat](#) \*A, const [INT](#) offset)

Re-index a *REAL* matrix in CSR format to make the index starting from 0 or 1.

- void [fasp\\_dcsr\\_symdiagscale](#) ([dCSRmat](#) \*A, const [dvector](#) \*diag)

Symmetric diagonal scaling  $D^{-1/2}AD^{-1/2}$ .

- [dCSRmat fasp\\_dcsr\\_sympart](#) ([dCSRmat](#) \*A)

Get symmetric part of a *dCSRmat* matrix.

- void [fasp\\_dcsr\\_multicoloring](#) ([dCSRmat](#) \*A, [INT](#) \*flags, [INT](#) \*groups)

Use the greedy multi-coloring to get color groups of the adjacency graph of *A*.

- void [fasp\\_dcsr\\_transz](#) ([dCSRmat](#) \*A, [INT](#) \*p, [dCSRmat](#) \*AT)

Generalized transpose of *A*: ( $n \times m$ ) matrix given in *dCSRmat* format.

- [dCSRmat fasp\\_dcsr\\_permz](#) ([dCSRmat](#) \*A, [INT](#) \*p)

Permute rows and cols of *A*, i.e.  $A = PAP'$  by the ordering in *p*.

- void [fasp\\_dcsr\\_sortz](#) ([dCSRmat](#) \*A, const [SHORT](#) isym)

Sort each row of *A* in ascending order w.r.t. column indices.

### 9.30.1 Detailed Description

Sparse matrix operations for [dCSRmat](#) matrices.

Note

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxSort.c](#), [AuxThreads.c](#), [AuxVector.c](#), and [BlaSpmvCSR.c](#)

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.30.2 Function Documentation

#### 9.30.2.1 fasp\_dcsr\_alloc()

```
void fasp_dcsr_alloc (
    const INT m,
    const INT n,
    const INT nnz,
    dCSRmat * A )
```

Allocate CSR sparse matrix memory space.

Parameters

<i>m</i>	Number of rows
----------	----------------

## Parameters

<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 134 of file `BlaSparseCSR.c`.

**9.30.2.2 fasp\_dcsr\_bandwidth()**

```
INT fasp_dcsr_bandwidth (
    const dCSRmat * A )
```

Get bandwidth of matrix.

## Parameters

<i>A</i>	pointer to the <a href="#">dCSRmat</a> matrix
----------	---

## Author

Zheng Li

## Date

03/22/2015

Definition at line 224 of file `BlaSparseCSR.c`.

**9.30.2.3 fasp\_dcsr\_compress()**

```
void fasp_dcsr_compress (
    const dCSRmat * A,
    dCSRmat * B,
    const REAL dtol )
```

Compress a CSR matrix A and store in CSR matrix B by dropping small entries  $\text{abs}(a_{ij}) \leq \text{dtol}$ .

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> CSR matrix
<i>B</i>	Pointer to <a href="#">dCSRmat</a> CSR matrix
<i>dtol</i>	Drop tolerance

**Author**

Shiquan Zhang

**Date**

03/10/2010

Modified by Chunsheng Feng, Zheng Li on 08/25/2012  
 Definition at line 1054 of file BlaSparseCSR.c.

**9.30.2.4 fasp\_dcsr\_compress\_inplace()**

```
SHORT fasp_dcsr_compress_inplace (
    dCSRmat * A,
    const REAL dtol )
```

Compress a CSR matrix A IN PLACE by dropping small entries  $\text{abs}(a_{ij}) \leq \text{dtol}$ .

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> CSR matrix
<i>dtol</i>	Drop tolerance

**Author**

Xiaozhe Hu

**Date**

12/25/2010

Modified by Chensong Zhang on 02/21/2013 Modified by Chunsheng Feng on 10/16/2020: Avoid filtering diagonal entries.

**Note**

This routine can be modified for filtering.

Definition at line 1134 of file BlaSparseCSR.c.

**9.30.2.5 fasp\_dcsr\_cp()**

```
void fasp_dcsr_cp (
    const dCSRmat * A,
    dCSRmat * B )
```

copy a [dCSRmat](#) to a new one B=A

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
<i>B</i>	Pointer to the <a href="#">dCSRmat</a> matrix

**Author**

Chensong Zhang

**Date**

04/06/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 822 of file BlaSparseCSR.c.

**9.30.2.6 fasp\_dcsr\_create()**

```
dCSRmat fasp_dcsr_create (
    const INT m,
    const INT n,
    const INT nnz )
```

Create CSR sparse matrix data memory space.

**Parameters**

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros

**Returns**

A the new dCSRmat matrix

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 43 of file BlaSparseCSR.c.

**9.30.2.7 fasp\_dcsr\_diagpref()**

```
void fasp_dcsr_diagpref (
    dCSRmat * A )
```

Re-order the column and data arrays of a CSR matrix, so that the first entry in each row is the diagonal.

**Parameters**

<i>A</i>	Pointer to the matrix to be re-ordered
----------	--

**Author**

Zhiyang Zhou



**Date**

09/09/2010

**Author**

Chunsheng Feng, Zheng Li

**Date**

09/02/2012

**Note**

Reordering is done in place.

Modified by Chensong Zhang on Dec/21/2012  
Definition at line 652 of file BlaSparseCSR.c.

**9.30.2.8 fasp\_dcsr\_free()**

```
void fasp_dcsr_free (
    dCSRmat * A )
```

Free CSR sparse matrix data memory space.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
----------	---

**Author**

Chensong Zhang

**Date**

2010/04/06 Modified by Chunsheng Feng on 08/11/2017: init A to NULL

Definition at line 177 of file BlaSparseCSR.c.

**9.30.2.9 fasp\_dcsr\_getblk()**

```
SHORT fasp_dcsr_getblk (
    const dCSRmat * A,
    const INT * Is,
    const INT * Js,
    const INT m,
    const INT n,
    dCSRmat * B )
```

Get a sub CSR matrix of A with specified rows and columns.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix
<i>B</i>	Pointer to <a href="#">dCSRmat</a> matrix

**Parameters**

<i>Is</i>	Pointer to selected rows
<i>Js</i>	Pointer to selected columns
<i>m</i>	Number of selected rows
<i>n</i>	Number of selected columns

**Returns**

FASP\_SUCCESS if succeeded, otherwise return error information.

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

12/25/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 423 of file BlaSparseCSR.c.

**9.30.2.10 fasp\_dcsr\_getcol()**

```
void fasp_dcsr_getcol (
    const INT n,
    const dCSRmat * A,
    REAL * col )
```

Get the n-th column of a CSR matrix A.

**Parameters**

<i>n</i>	Index of a column of A (0 <= n <= A.col-1)
<i>A</i>	Pointer to <a href="#">dCSRmat</a> CSR matrix
<i>col</i>	Pointer to the column

**Author**

Xiaozhe Hu

**Date**

11/07/2009

Modified by Chunsheng Feng, Zheng Li on 07/08/2012  
Definition at line 573 of file BlaSparseCSR.c.

**9.30.2.11 fasp\_dcsr\_getdiag()**

```
void fasp_dcsr_getdiag (
    INT n,
    const dCSRmat * A,
    dvector * diag )
```

Get first  $n$  diagonal entries of a CSR matrix  $A$ .

## Parameters

<i>n</i>	Number of diagonal entries to get (if $n=0$ , then get all diagonal entries)
<i>A</i>	Pointer to <a href="#">dCSRmat</a> CSR matrix
<i>diag</i>	Pointer to the diagonal as a dvector

## Author

Chensong Zhang

## Date

05/20/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
 Definition at line 509 of file BlaSparseCSR.c.

**9.30.2.12 fasp\_dcsr\_multicoloring()**

```
void fasp_dcsr_multicoloring (
    dCSRmat * A,
    INT * flags,
    INT * groups )
```

Use the greedy multi-coloring to get color groups of the adjacency graph of A.

## Parameters

<i>A</i>	Input <a href="#">dCSRmat</a>
<i>flags</i>	flags for the independent group
<i>groups</i>	Return group numbers

## Author

Chunsheng Feng

## Date

09/15/2012

Definition at line 1362 of file BlaSparseCSR.c.

**9.30.2.13 fasp\_dcsr\_perm()**

```
dCSRmat fasp_dcsr_perm (
    dCSRmat * A,
    INT * P )
```

Apply permutation of A, i.e.  $A_{perm}=PAP'$  by the orders given in P.

## Parameters

<i>A</i>	Pointer to the original <a href="#">dCSRmat</a> matrix
<i>P</i>	Pointer to orders

**Returns**

The new ordered [dCSRmat](#) matrix if succeed, NULL if fail

**Author**

Shiquan Zhang

**Date**

03/10/2010

**Note**

$P[i] = k$  means k-th row and column become i-th row and column!

Deprecated! Will be replaced by `fasp_dcsr_permz` later. –Chensong

Modified by Chunsheng Feng, Zheng Li on 07/12/2012

Definition at line 254 of file BlaSparseCSR.c.

**9.30.2.14 fasp\_dcsr\_permz()**

```
dCSRmat fasp_dcsr_permz (
    dCSRmat * A,
    INT * p )
```

Permute rows and cols of A, i.e.  $A=PAP'$  by the ordering in p.

**Parameters**

<i>A</i>	Pointer to the original <a href="#">dCSRmat</a> matrix
<i>p</i>	Pointer to ordering

**Note**

This is just applying twice `fasp_dcsr_transz(&A,p,At)`.

In matlab notation:  $A_{perm}=A(p,p)$ ;

**Returns**

The new ordered [dCSRmat](#) matrix if succeed, NULL if fail

**Author**

Ludmil Zikatanov

**Date**

19951219 (Fortran), 20150912 (C)

Definition at line 1583 of file BlaSparseCSR.c.

### 9.30.2.15 fasp\_dcsr\_regdiag()

```
SHORT fasp_dcsr_regdiag (  
    dCSRmat * A,  
    const REAL value )
```

Regularize diagonal entries of a CSR sparse matrix.

## Parameters

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
<i>value</i>	Set a value on diag(A) which is too close to zero to "value"

## Returns

FASP\_SUCCESS if no diagonal entry is close to zero, else ERROR

## Author

Shiquan Zhang

## Date

11/07/2009

Definition at line 758 of file BlaSparseCSR.c.

**9.30.2.16 fasp\_dcsr\_shift()**

```
void fasp_dcsr_shift (
    dCSRmat * A,
    const INT offset )
```

Re-index a REAL matrix in CSR format to make the index starting from 0 or 1.

## Parameters

<i>A</i>	Pointer to CSR matrix
<i>offset</i>	Size of offset (1 or -1)

## Author

Chensong Zhang

## Date

04/06/2010

Modified by Chunsheng Feng, Zheng Li on 07/11/2012

Definition at line 1181 of file BlaSparseCSR.c.

**9.30.2.17 fasp\_dcsr\_sort()**

```
void fasp_dcsr_sort (
    dCSRmat * A )
```

Sort each row of A in ascending order w.r.t. column indices.

## Parameters

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
----------	---

**Author**

Shiquan Zhang

**Date**

06/10/2010

Definition at line 365 of file BlaSparseCSR.c.

**9.30.2.18 fasp\_dcsr\_sortz()**

```
void fasp_dcsr_sortz (
    dCSRmat * A,
    const SHORT isym )
```

Sort each row of A in ascending order w.r.t. column indices.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
<i>isym</i>	Flag for symmetry, =[0/nonzero]=[general/symmetric] matrix

**Note**

Applying twice [fasp\\_dcsr\\_transz\(\)](#), if A is symmetric, then the transpose is applied only once and then AT copied on A.

**Author**

Ludmil Zikatanov

**Date**

19951219 (Fortran), 20150912 (C)

Definition at line 1615 of file BlaSparseCSR.c.

**9.30.2.19 fasp\_dcsr\_symdiagscale()**

```
void fasp_dcsr_symdiagscale (
    dCSRmat * A,
    const dvector * diag )
```

Symmetric diagonal scaling  $D^{-1/2}AD^{-1/2}$ .**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
<i>diag</i>	Pointer to the diagonal entries

**Author**

Xiaozhe Hu



## Date

01/31/2011

Modified by Chunsheng Feng, Zheng Li on 07/11/2012  
Definition at line 1242 of file BlasparsedCSR.c.

**9.30.2.20 fasp\_dcsr\_sympart()**

```
dCSRmat fasp_dcsr_sympart (  
    dCSRmat * A )
```

Get symmetric part of a **dCSRmat** matrix.

## Parameters

<b>A</b>	Pointer to the <b>dCSRmat</b> matrix
----------	--------------------------------------

## Returns

Symmetrized the **dCSRmat** matrix

## Author

Xiaozhe Hu

## Date

03/21/2011

Definition at line 1329 of file BlasparsedCSR.c.

**9.30.2.21 fasp\_dcsr\_trans()**

```
void fasp_dcsr_trans (  
    const dCSRmat * A,  
    dCSRmat * AT )
```

Find transpose of **dCSRmat** matrix A.

## Parameters

<b>A</b>	Pointer to the <b>dCSRmat</b> matrix
<b>AT</b>	Pointer to the transpose of <b>dCSRmat</b> matrix A (output)

## Author

Chensong Zhang

## Date

04/06/2010

Modified by Chunsheng Feng, Zheng Li on 06/20/2012  
Definition at line 923 of file BlasparsedCSR.c.

### 9.30.2.22 fasp\_dcsr\_transpose()

```
void fasp_dcsr_transpose (
    INT * row[2],
    INT * col[2],
    REAL * val[2],
    INT * nn,
    INT * tniz )
```

Transpose of a [dCSRmat](#) matrix.

#### Note

This subroutine transpose in CSR format IN ORDER

#### Parameters

<i>row</i>	Pointers of the rows of the matrix and its transpose
<i>col</i>	Pointers of the columns of the matrix and its transpose
<i>val</i>	Pointers to the values of the matrix and its transpose
<i>nn</i>	Pointer to the number of rows/columns of A and A'
<i>tniz</i>	Pointer to the number of nonzeros A and A'

#### Author

Shuo Zhang

#### Date

07/06/2009

Definition at line 1003 of file BlaSparseCSR.c.

### 9.30.2.23 fasp\_dcsr\_transz()

```
void fasp_dcsr_transz (
    dCSRmat * A,
    INT * p,
    dCSRmat * AT )
```

Generalized transpose of A: (n x m) matrix given in [dCSRmat](#) format.

#### Parameters

<i>A</i>	Pointer to matrix in <a href="#">dCSRmat</a> for transpose, INPUT
<i>p</i>	Permutation, INPUT
<i>AT</i>	Pointer to matrix AT = transpose(A) if p = NULL, OR AT = transpose(A)p if p is not NULL

#### Note

The storage for all pointers in AT should already be allocated, i.e. AT->IA, AT->JA and AT->val should be allocated before calling this function. If A.val=NULL, then AT->val[] is not changed.

performs  $AT = \text{transpose}(A)p$ , where p is a permutation. If p=NULL then p=I is assumed. Applying twice this procedure one gets  $AT = \text{transpose}(\text{transpose}(A)p)p = \text{transpose}(p)Ap$ , which is the same A with rows and columns permuted according to p.

If A=NULL, then only transposes/permutates the structure of A.

For p=NULL, applying this two times  $A \rightarrow A^T \rightarrow A$  orders all the row indices in A in increasing order.

Reference: Fred G. Gustavson. Two fast algorithms for sparse matrices: multiplication and permuted transposition. ACM Trans. Math. Software, 4(3):250-269, 1978.

#### Author

Ludmil Zikatanov

#### Date

19951219 (Fortran), 20150912 (C)

Definition at line 1463 of file BlaSparseCSR.c.

### 9.30.2.24 fasp\_icsr\_cp()

```
void fasp_icsr_cp (
    const iCSRmat * A,
    iCSRmat * B )
```

Copy a [iCSRmat](#) to a new one B=A.

#### Parameters

<i>A</i>	Pointer to the <a href="#">iCSRmat</a> matrix
<i>B</i>	Pointer to the <a href="#">iCSRmat</a> matrix

#### Author

Chensong Zhang

#### Date

05/16/2013

Definition at line 797 of file BlaSparseCSR.c.

### 9.30.2.25 fasp\_icsr\_create()

```
iCSRmat fasp_icsr_create (
    const INT m,
    const INT n,
    const INT nnz )
```

Create CSR sparse matrix data memory space.

#### Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros

**Returns**

A the new [iCSRmat](#) matrix

**Author**

Chensong Zhang

**Date**

2010/04/06

Definition at line 89 of file BlaSparseCSR.c.

**9.30.2.26 fasp\_icsr\_free()**

```
void fasp_icsr_free (
    iCSRmat * A )
```

Free CSR sparse matrix data memory space.

**Parameters**

<i>A</i>	Pointer to the <a href="#">iCSRmat</a> matrix
----------	---

**Author**

Chensong Zhang

**Date**

2010/04/06 Modified by Chunsheng Feng on 08/11/2017: init A to NULL

Definition at line 201 of file BlaSparseCSR.c.

**9.30.2.27 fasp\_icsr\_trans()**

```
void fasp_icsr_trans (
    const iCSRmat * A,
    iCSRmat * AT )
```

Find transpose of [iCSRmat](#) matrix A.

**Parameters**

<i>A</i>	Pointer to the <a href="#">iCSRmat</a> matrix A
<i>AT</i>	Pointer to the <a href="#">iCSRmat</a> matrix A'

**Author**

Chensong Zhang

Date

04/06/2010

Modified by Chunsheng Feng, Zheng Li on 06/20/2012  
Definition at line 847 of file BlaSparseCSR.c.

## 9.31 BlaSparseCSR.c File Reference

Sparse matrix operations for [dCSRMat](#) matrices.

```
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [dCSRMat](#) \* [fasp\\_dcsr\\_create](#) (const [INT](#) num\_rows, const [INT](#) num\_cols, const [INT](#) num\_nonzeros)  
*Create a [dCSRMat](#) object.*
- void [fasp\\_dcsr\\_free](#) ([dCSRMat](#) \*A)  
*Destroy a [dCSRMat](#) object.*

#### 9.31.1 Detailed Description

Sparse matrix operations for [dCSRMat](#) matrices.

Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#)

Reference: John Mellor-Crummey and John Garvin Optimizaing sparse matrix vector product computations using unroll and jam, Tech Report Rice Univ, Aug 2002.  
Copyright (C) 2011–2020 by the FASP team. All rights reserved.

---

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.31.2 Function Documentation

##### 9.31.2.1 [fasp\\_dcsr\\_create\(\)](#)

```
dCSRMat * fasp\_dcsr\_create (
    const INT num_rows,
    const INT num_cols,
    const INT num_nonzeros )
```

Create a [dCSRMat](#) object.

Parameters

<i>num_rows</i>	Number of rows
<i>num_cols</i>	Number of cols
<i>num_nonzeros</i>	Number of nonzero entries

**Author**

Zhiyang Zhou

**Date**

01/07/2011

Definition at line 39 of file BlaSparseCSR.c.

**9.31.2.2 fasp\_dcsr\_free()**

```
void fasp_dcsr_free (
    dCSRmat * A )
```

Destroy a [dCSRmat](#) object.**Parameters**

<b>A</b>	Pointer to the <a href="#">dCSRmat</a> type matrix
----------	--

**Author**

Zhiyang Zhou

**Date**

01/07/2011

Definition at line 67 of file BlaSparseCSR.c.

**9.32 BlaSparseSTR.c File Reference**Sparse matrix operations for [dSTRmat](#) matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- [dSTRmat fasp\\_dstr\\_create](#) (const [INT](#) nx, const [INT](#) ny, const [INT](#) nz, const [INT](#) nc, const [INT](#) nband, [INT](#) \*offsets)  
*Create STR sparse matrix data memory space.*
- void [fasp\\_dstr\\_alloc](#) (const [INT](#) nx, const [INT](#) ny, const [INT](#) nz, const [INT](#) nxy, const [INT](#) ngrid, const [INT](#) nband, const [INT](#) nc, [INT](#) \*offsets, [dSTRmat](#) \*A)  
*Allocate STR sparse matrix memory space.*
- void [fasp\\_dstr\\_free](#) ([dSTRmat](#) \*A)  
*Free STR sparse matrix data memeory space.*
- void [fasp\\_dstr\\_cp](#) (const [dSTRmat](#) \*A, [dSTRmat](#) \*B)  
*Copy a [dSTRmat](#) to a new one B=A.*

### 9.32.1 Detailed Description

Sparse matrix operations for [dSTRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.32.2 Function Documentation

#### 9.32.2.1 fasp\_dstr\_alloc()

```
void fasp_dstr_alloc (
    const INT nx,
    const INT ny,
    const INT nz,
    const INT nxy,
    const INT ngrid,
    const INT nband,
    const INT nc,
    INT * offsets,
    dSTRmat * A )
```

Allocate STR sparse matrix memory space.

#### Parameters

<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>nz</i>	Number of grids in z direction
<i>nxy</i>	Number of grids in x-y plane
<i>ngrid</i>	Number of grids
<i>nband</i>	Number of off-diagonal bands
<i>nc</i>	Number of components
<i>offsets</i>	Shift from diagonal
<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix

#### Author

Shiquan Zhang, Xiaozhe Hu

#### Date

05/17/2010

Definition at line 93 of file BlaSparseSTR.c.

### 9.32.2.2 fasp\_dstr\_cp()

```
void fasp_dstr_cp (
    const dSTRmat * A,
    dSTRmat * B )
```

Copy a `dSTRmat` to a new one `B=A`.

#### Parameters

<i>A</i>	Pointer to the <code>dSTRmat</code> matrix
<i>B</i>	Pointer to the <code>dSTRmat</code> matrix

#### Author

Zhiyang Zhou

#### Date

04/21/2010

Definition at line 162 of file `BlaSparseSTR.c`.

### 9.32.2.3 fasp\_dstr\_create()

```
dSTRmat fasp_dstr_create (
    const INT nx,
    const INT ny,
    const INT nz,
    const INT nc,
    const INT nband,
    INT * offsets )
```

Create STR sparse matrix data memory space.

#### Parameters

<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>nz</i>	Number of grids in z direction
<i>nc</i>	Number of components
<i>nband</i>	Number of off-diagonal bands
<i>offsets</i>	Shift from diagonal

#### Returns

The `dSTRmat` matrix

#### Author

Shiquan Zhang, Xiaozhe Hu



Date

05/17/2010

Definition at line 41 of file BlaSparseSTR.c.

#### 9.32.2.4 fasp\_dstr\_free()

```
void fasp_dstr_free (
    dSTRmat * A )
```

Free STR sparse matrix data memeory space.

Parameters

A	Pointer to the dSTRmat matrix
---	-------------------------------

Author

Shiquan Zhang, Xiaozhe Hu

Date

05/17/2010

Definition at line 136 of file BlaSparseSTR.c.

## 9.33 BlaSparseUtil.c File Reference

Routines for sparse matrix operations.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void `fasp_sparse_abybms_` (INT \*ia, INT \*ja, INT \*ib, INT \*jb, INT \*nap, INT \*map, INT \*mbp, INT \*ic, INT \*jc)  
*Multiplication of two sparse matrices: calculating the nonzero structure of the result if jc is not null. If jc is null only finds num of non zeroes.*
- void `fasp_sparse_abyb_` (INT \*ia, INT \*ja, REAL \*a, INT \*ib, INT \*jb, REAL \*b, INT \*nap, INT \*map, INT \*mbp, INT \*ic, INT \*jc, REAL \*c)  
*Multiplication of two sparse matrices.*
- void `fasp_sparse_iit_` (INT \*ia, INT \*ja, INT \*na, INT \*ma, INT \*iat, INT \*jat)  
*Transpose a boolean matrix (only given by ia, ja)*
- void `fasp_sparse_aat_` (INT \*ia, INT \*ja, REAL \*a, INT \*na, INT \*ma, INT \*iat, INT \*jat, REAL \*at)  
*Transpose a boolean matrix (only given by ia, ja)*
- void `fasp_sparse_aplbms_` (INT \*ia, INT \*ja, INT \*ib, INT \*jb, INT \*nab, INT \*mab, INT \*ic, INT \*jc)  
*Addition of two sparse matrices: calculating the nonzero structure of the result if jc is not null. if jc is null only finds num of non zeroes.*
- void `fasp_sparse_aplusb_` (INT \*ia, INT \*ja, REAL \*a, INT \*ib, INT \*jb, REAL \*b, INT \*nab, INT \*mab, INT \*ic, INT \*jc, REAL \*c)

*Addition of two sparse matrices.*

- void `fasp_sparse_rapms_` (INT \*ir, INT \*jr, INT \*ia, INT \*ja, INT \*ip, INT \*jp, INT \*nin, INT \*ncin, INT \*iac, INT \*jac, INT \*maxrout)

*Calculates the nonzero structure of  $R \cdot A \cdot P$ , if jac is not null. If jac is null only finds num of nonzeros.*

- void `fasp_sparse_wtams_` (INT \*jw, INT \*ia, INT \*ja, INT \*nwp, INT \*map, INT \*jv, INT \*nvp, INT \*icp)

*Finds the nonzeros in the result of  $v^t = w^t A$ , where w is a sparse vector and A is sparse matrix. jv is an integer array containing the indices of the nonzero elements in the result.*

- void `fasp_sparse_wta_` (INT \*jw, REAL \*w, INT \*ia, INT \*ja, REAL \*a, INT \*nwp, INT \*map, INT \*jv, REAL \*v, INT \*nvp)

*Calculate  $v^t = w^t A$ , where w is a sparse vector and A is sparse matrix. v is an array of dimension = number of columns in A.*

- void `fasp_sparse_ytxbig_` (INT \*jy, REAL \*y, INT \*nyp, REAL \*x, REAL \*s)

*Calculates  $s = y^t x$ . y-sparse, x - no.*

- void `fasp_sparse_ytx_` (INT \*jy, REAL \*y, INT \*jx, REAL \*x, INT \*nyp, INT \*npx, INT \*icp, REAL \*s)

*Calculates  $s = y^t x$ . y is sparse, x is sparse.*

- void `fasp_sparse_rapcmp_` (INT \*ir, INT \*jr, REAL \*r, INT \*ia, INT \*ja, REAL \*a, INT \*ipt, INT \*jpt, REAL \*pt, INT \*nin, INT \*ncin, INT \*iac, INT \*jac, REAL \*ac, INT \*idummy)

*Calculates  $R \cdot A \cdot P$  after the nonzero structure of the result is known. iac,jac,ac have to be allocated before call to this function.*

- `ivector fasp_sparse_mis` (dCSRmat \*A)

*Get the maximal independent set of a CSR matrix.*

### 9.33.1 Detailed Description

Routines for sparse matrix operations.

Note

Most algorithms work as follows: (a) Boolean operations (to determine the nonzero structure); (b) Numerical part, where the result is calculated.

Parameter notation :I: is input; :O: is output; :IO: is both

This file contains Level-1 (Bla) functions. It requires: [AuxMemory.c](#)

---

Copyright (C) 2010–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.33.2 Function Documentation

#### 9.33.2.1 fasp\_sparse\_aat\_()

```
void fasp_sparse_aat_ (
    INT * ia,
    INT * ja,
    REAL * a,
    INT * na,
    INT * ma,
    INT * iat,
    INT * jat,
    REAL * at )
```

Transpose a boolean matrix (only given by ia, ja)

## Parameters

<i>ia</i>	array of row pointers (as usual in CSR)
<i>ja</i>	array of column indices
<i>a</i>	array of entries of teh input
<i>na</i>	number of rows of A
<i>ma</i>	number of cols of A
<i>iat</i>	array of row pointers in the result
<i>jat</i>	array of column indices
<i>at</i>	array of entries of the result

Definition at line 273 of file BlaSparseUtil.c.

## 9.33.2.2 fasp\_sparse\_abyb\_()

```
void fasp_sparse_abyb_ (
    INT * ia,
    INT * ja,
    REAL * a,
    INT * ib,
    INT * jb,
    REAL * b,
    INT * nap,
    INT * map,
    INT * mbp,
    INT * ic,
    INT * jc,
    REAL * c )
```

Multiplication of two sparse matrices.

## Parameters

<i>ia</i>	array of row pointers 1st multiplicand
<i>ja</i>	array of column indices 1st multiplicand
<i>a</i>	entries of the 1st multiplicand
<i>ib</i>	array of row pointers 2nd multiplicand
<i>jb</i>	array of column indices 2nd multiplicand
<i>b</i>	entries of the 2nd multiplicand
<i>ic</i>	array of row pointers in c=a*b
<i>jc</i>	array of column indices in c=a*b
<i>c</i>	entries of the result: c= a*b
<i>nap</i>	number of rows in the 1st multiplicand
<i>map</i>	number of columns in the 1st multiplicand
<i>mbp</i>	number of columns in the 2nd multiplicand

Modified by Chensong Zhang on 09/11/2012

Definition at line 127 of file BlaSparseUtil.c.

### 9.33.2.3 fasp\_sparse\_abybms\_()

```
void fasp_sparse_abybms_ (
    INT * ia,
    INT * ja,
    INT * ib,
    INT * jb,
    INT * nap,
    INT * map,
    INT * mbp,
    INT * ic,
    INT * jc )
```

Multiplication of two sparse matrices: calculating the nonzero structure of the result if jc is not null. If jc is null only finds num of nonzeros.

#### Parameters

<i>ia</i>	array of row pointers 1st multiplicand
<i>ja</i>	array of column indices 1st multiplicand
<i>ib</i>	array of row pointers 2nd multiplicand
<i>jb</i>	array of column indices 2nd multiplicand
<i>nap</i>	number of rows of A
<i>map</i>	number of cols of A
<i>mbp</i>	number of cols of b
<i>ic</i>	array of row pointers in the result (this is also computed here again, so that we can have a stand alone call of this routine, if for some reason the number of nonzeros in the result is known)
<i>jc</i>	array of column indices in the result c=a*b

Modified by Chensong Zhang on 09/11/2012  
Definition at line 52 of file BlaSparseUtil.c.

### 9.33.2.4 fasp\_sparse\_aplbms\_()

```
void void fasp_sparse_aplbms_ (
    INT * ia,
    INT * ja,
    INT * ib,
    INT * jb,
    INT * nab,
    INT * mab,
    INT * ic,
    INT * jc )
```

Addition of two sparse matrices: calculating the nonzero structure of the result if jc is not null. if jc is null only finds num of nonzeros.

#### Parameters

<i>ia</i>	array of row pointers 1st summand
<i>ja</i>	array of column indices 1st summand
<i>ib</i>	array of row pointers 2nd summand
<i>jb</i>	array of column indices 2nd summand
<i>nab</i>	number of rows

## Parameters

<i>mab</i>	number of cols
<i>ic</i>	array of row pointers in the result (this is also computed here again, so that we can have a stand alone call of this routine, if for some reason the number of nonzeros in the result is known)
<i>jc</i>	array of column indices in the result $c=a+b$

Definition at line 359 of file BlasSparseUtil.c.

## 9.33.2.5 fasp\_sparse\_aplusb\_()

```
void fasp_sparse_aplusb_ (
    INT * ia,
    INT * ja,
    REAL * a,
    INT * ib,
    INT * jb,
    REAL * b,
    INT * nab,
    INT * mab,
    INT * ic,
    INT * jc,
    REAL * c )
```

Addition of two sparse matrices.

## Parameters

<i>ia</i>	array of row pointers 1st summand
<i>ja</i>	array of column indices 1st summand
<i>a</i>	entries of the 1st summand
<i>ib</i>	array of row pointers 2nd summand
<i>jb</i>	array of column indices 2nd summand
<i>b</i>	entries of the 2nd summand
<i>nab</i>	number of rows
<i>mab</i>	number of cols
<i>ic</i>	array of row pointers in $c=a+b$
<i>jc</i>	array of column indices in $c=a+b$
<i>c</i>	entries of the result: $c=a+b$

Definition at line 431 of file BlasSparseUtil.c.

## 9.33.2.6 fasp\_sparse\_iit\_()

```
void fasp_sparse_iit_ (
    INT * ia,
    INT * ja,
    INT * na,
    INT * ma,
    INT * iat,
    INT * jat )
```

Transpose a boolean matrix (only given by ia, ja)

#### Parameters

<i>ia</i>	array of row pointers (as usual in CSR)
<i>ja</i>	array of column indices
<i>na</i>	number of rows
<i>ma</i>	number of cols
<i>iat</i>	array of row pointers in the result
<i>jat</i>	array of column indices

Definition at line 197 of file BlaSparseUtil.c.

#### 9.33.2.7 fasp\_sparse\_mis()

```
ivector fasp_sparse_mis (
    dCSRmat * A )
```

Get the maximal independet set of a CSR matrix.

#### Parameters

<i>A</i>	pointer to the matrix
----------	-----------------------

#### Note

Only use the sparsity of A, index starts from 1 (fortran)!!

Definition at line 907 of file BlaSparseUtil.c.

#### 9.33.2.8 fasp\_sparse\_rapcmp\_()

```
void fasp_sparse_rapcmp_ (
    INT * ir,
    INT * jr,
    REAL * r,
    INT * ia,
    INT * ja,
    REAL * a,
    INT * ipt,
    INT * jpt,
    REAL * pt,
    INT * nin,
    INT * ncin,
    INT * iac,
    INT * jac,
    REAL * ac,
    INT * idummy )
```

Calculates  $R \cdot A \cdot P$  after the nonzero structure of the result is known. iac,jac,ac have to be allocated before call to this function.

**Note**

:I: is input :O: is output :IO: is both

**Parameters**

<i>ir</i>	:I: array of row pointers for R
<i>jr</i>	:I: array of column indices for R
<i>r</i>	:I: entries of R
<i>ia</i>	:I: array of row pointers for A
<i>ja</i>	:I: array of column indices for A
<i>a</i>	:I: entries of A
<i>ipt</i>	:I: array of row pointers for P
<i>jpt</i>	:I: array of column indices for P
<i>pt</i>	:I: entries of P
<i>nin</i>	:I: number of rows in R
<i>ncin</i>	:I: number of rows in
<i>iac</i>	:O: array of row pointers for P
<i>jac</i>	:O: array of column indices for P
<i>ac</i>	:O: entries of P
<i>idummy</i>	not changed

**Note**

Compute  $R \cdot A \cdot P$  for known nonzero structure of the result the result is stored in iac,jac,ac!

Definition at line 787 of file BlasparseUtil.c.

**9.33.2.9 fasp\_sparse\_rapms\_()**

```
void fasp_sparse_rapms_ (
    INT * ir,
    INT * jr,
    INT * ia,
    INT * ja,
    INT * ip,
    INT * jp,
    INT * nin,
    INT * ncin,
    INT * iac,
    INT * jac,
    INT * maxrout )
```

Calculates the nonzero structure of  $R \cdot A \cdot P$ , if jac is not null. If jac is null only finds num of nonzeros.

**Note**

:I: is input :O: is output :IO: is both

**Parameters**

<i>ir</i>	:I: array of row pointers for R
-----------	---------------------------------

## Parameters

<i>jr</i>	:I: array of column indices for R
<i>ia</i>	:I: array of row pointers for A
<i>ja</i>	:I: array of column indices for A
<i>ip</i>	:I: array of row pointers for P
<i>jp</i>	:I: array of column indices for P
<i>nin</i>	:I: number of rows in R
<i>ncin</i>	:I: number of columns in R
<i>iac</i>	:O: array of row pointers for Ac
<i>jac</i>	:O: array of column indices for Ac
<i>maxrout</i>	:O: the maximum nonzeros per row for R

## Note

Computes the sparsity pattern of  $R \cdot A \cdot P$ . maxrout is output and is the maximum nonzeros per row for r. On output we also have iac (if jac is null) and jac (if jac entry is not null). R is (nc,n) A is (n,n) and P is (n,nc)!

Modified by Chensong Zhang on 09/11/2012

Definition at line 515 of file BlaSparseUtil.c.

## 9.33.2.10 fasp\_sparse\_wta\_()

```
void fasp_sparse_wta_ (
    INT * jw,
    REAL * w,
    INT * ia,
    INT * ja,
    REAL * a,
    INT * nwp,
    INT * map,
    INT * jv,
    REAL * v,
    INT * nvp )
```

Calculate  $v^t = w^t A$ , where w is a sparse vector and A is sparse matrix. v is an array of dimension = number of columns in A.

## Note

:I: is input :O: is output :IO: is both

## Parameters

<i>jw</i>	:I: indices such that w[jw] is nonzero
<i>w</i>	:I: the values of w
<i>ia</i>	:I: array of row pointers for A
<i>ja</i>	:I: array of column indices for A
<i>a</i>	:I: entries of A
<i>nwp</i>	:I: number of nonzeros in w (the length of w)
<i>map</i>	:I: number of columns in A
<i>jv</i>	:O: indices such that v[jv] is nonzero



## Parameters

<i>v</i>	:O: the result $v^t = w^t A$
<i>nvp</i>	:I: number of nonzeros in <i>v</i>

Definition at line 648 of file BlaSparseUtil.c.

**9.33.2.11 fasp\_sparse\_wtams\_()**

```
void fasp_sparse_wtams_ (
    INT * jw,
    INT * ia,
    INT * ja,
    INT * nwp,
    INT * map,
    INT * jv,
    INT * nvp,
    INT * icp )
```

Finds the nonzeros in the result of  $v^t = w^t A$ , where *w* is a sparse vector and *A* is sparse matrix. *jv* is an integer array containing the indices of the nonzero elements in the result.

:I: is input :O: is output :IO: is both

## Parameters

<i>jw</i>	:I: indices such that <i>w</i> [ <i>jw</i> ] is nonzero
<i>ia</i>	:I: array of row pointers for <i>A</i>
<i>ja</i>	:I: array of column indices for <i>A</i>
<i>nwp</i>	:I: number of nonzeros in <i>w</i> (the length of <i>w</i> )
<i>map</i>	:I: number of columns in <i>A</i>
<i>jv</i>	:O: indices such that <i>v</i> [ <i>jv</i> ] is nonzero
<i>nvp</i>	:I: number of nonzeros in <i>v</i>
<i>icp</i>	:IO: is a working array of length (* <i>map</i> ) which on output satisfies <i>icp</i> [ <i>jv</i> [ <i>k</i> ]-1]= <i>k</i> ; Values of <i>icp</i> [] at positions * other than ( <i>jv</i> [ <i>k</i> ]-1) remain unchanged.

Modified by Chensong Zhang on 09/11/2012

Definition at line 596 of file BlaSparseUtil.c.

**9.33.2.12 fasp\_sparse\_ytx\_()**

```
void fasp_sparse_ytx_ (
    INT * jy,
    REAL * y,
    INT * jx,
    REAL * x,
    INT * nyp,
    INT * nxp,
    INT * icp,
    REAL * s )
```

Calculates  $s = y^t x$ . *y* is sparse, *x* is sparse.

**Note**

:I: is input :O: is output :IO: is both

**Parameters**

<i>jy</i>	:I: indices such that $y[jy]$ is nonzero
<i>y</i>	:I: is a sparse vector.
<i>nyp</i>	:I: number of non zeroes in $y$
<i>jx</i>	:I: indices such that $x[jx]$ is nonzero
<i>x</i>	:I: is a sparse vector.
<i>nxp</i>	:I: number of non zeroes in $x$
<i>icp</i>	???
<i>s</i>	:O: $s = y^t x$ .

Definition at line 733 of file BlaSparseUtil.c.

**9.33.2.13 fasp\_sparse\_ytxbig\_()**

```
void fasp_sparse_ytxbig_ (
    INT * jy,
    REAL * y,
    INT * nyp,
    REAL * x,
    REAL * s )
```

Calculates  $s = y^t x$ .  $y$ -sparse,  $x$  - no.

**Note**

:I: is input :O: is output :IO: is both

**Parameters**

<i>jy</i>	:I: indices such that $y[jy]$ is nonzero
<i>y</i>	:I: is a sparse vector
<i>nyp</i>	:I: number of non zeroes in $v$
<i>x</i>	:I: also a vector assumed to have entry for any $j=jy[i]-1$ ; for $i=1:nyp$ . This means that $x$ here does not have to be sparse
<i>s</i>	:O: $s = y^t x$

Definition at line 699 of file BlaSparseUtil.c.

**9.34 BLASpmvBLC.c File Reference**

Linear algebraic operations for [dBLMat](#) matrices.

```
#include <time.h>
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
```

## Functions

- void `fasp_blas_dblc_aApy` (const `REAL` `alpha`, const `dBLCmat` `*A`, const `REAL` `*x`, `REAL` `*y`)  
*Matrix-vector multiplication  $y = \text{alpha} * A * x + y$ .*
- void `fasp_blas_dblc_mxv` (const `dBLCmat` `*A`, const `REAL` `*x`, `REAL` `*y`)  
*Matrix-vector multiplication  $y = A * x$ .*

### 9.34.1 Detailed Description

Linear algebraic operations for `dBLCmat` matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [BlaSpmvCSR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.34.2 Function Documentation

#### 9.34.2.1 `fasp_blas_dblc_aApy()`

```
void fasp_blas_dblc_aApy (
    const REAL alpha,
    const dBLCmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = \text{alpha} * A * x + y$ .

#### Parameters

<i>alpha</i>	REAL factor a
<i>A</i>	Pointer to <code>dBLCmat</code> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

#### Author

Xiaozhe Hu

#### Date

06/04/2010

Definition at line 38 of file BlasSpmvBLC.c.

#### 9.34.2.2 `fasp_blas_dblc_mxv()`

```
void fasp_blas_dblc_mxv (
    const dBLCmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = A * x$ .

## Parameters

<i>A</i>	Pointer to <a href="#">dBLCmat</a> matrix <i>A</i>
<i>x</i>	Pointer to array <i>x</i>
<i>y</i>	Pointer to array <i>y</i>

## Author

Chensong Zhang

## Date

04/27/2013

Definition at line 164 of file `BlaSpmvBLC.c`.

## 9.35 BlaSpmvBSR.c File Reference

Linear algebraic operations for [dBSRmat](#) matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_blas\\_dbsr\\_axm](#) ([dBSRmat](#) \**A*, const [REAL](#) *alpha*)  
*Multiply a sparse matrix A in BSR format by a scalar alpha.*
- void [fasp\\_blas\\_dbsr\\_aAxpby](#) (const [REAL](#) *alpha*, [dBSRmat](#) \**A*, [REAL](#) \**x*, const [REAL](#) *beta*, [REAL](#) \**y*)  
*Compute  $y := \alpha * A * x + \beta * y$ .*
- void [fasp\\_blas\\_dbsr\\_aAxy](#) (const [REAL](#) *alpha*, const [dBSRmat](#) \**A*, const [REAL](#) \**x*, [REAL](#) \**y*)  
*Compute  $y := \alpha * A * x + y$ .*
- void [fasp\\_blas\\_dbsr\\_aAxy\\_agg](#) (const [REAL](#) *alpha*, const [dBSRmat](#) \**A*, const [REAL](#) \**x*, [REAL](#) \**y*)  
*Compute  $y := \alpha * A * x + y$  where each small block matrix is an identity matrix.*
- void [fasp\\_blas\\_dbsr\\_mxv](#) (const [dBSRmat](#) \**A*, const [REAL](#) \**x*, [REAL](#) \**y*)  
*Compute  $y := A * x$ .*
- void [fasp\\_blas\\_dbsr\\_mxv\\_agg](#) (const [dBSRmat](#) \**A*, const [REAL](#) \**x*, [REAL](#) \**y*)  
*Compute  $y := A * x$ , where each small block matrices of A is an identity.*
- void [fasp\\_blas\\_dbsr\\_mxm](#) (const [dBSRmat](#) \**A*, const [dBSRmat](#) \**B*, [dBSRmat](#) \**C*)  
*Sparse matrix multiplication  $C = A * B$ .*
- void [fasp\\_blas\\_dbsr\\_rap1](#) (const [dBSRmat](#) \**R*, const [dBSRmat](#) \**A*, const [dBSRmat](#) \**P*, [dBSRmat](#) \**B*)  
*dBSRmat sparse matrix multiplication  $B = R * A * P$*
- void [fasp\\_blas\\_dbsr\\_rap](#) (const [dBSRmat](#) \**R*, const [dBSRmat](#) \**A*, const [dBSRmat](#) \**P*, [dBSRmat](#) \**B*)  
*dBSRmat sparse matrix multiplication  $B = R * A * P$*
- void [fasp\\_blas\\_dbsr\\_rap\\_agg](#) (const [dBSRmat](#) \**R*, const [dBSRmat](#) \**A*, const [dBSRmat](#) \**P*, [dBSRmat](#) \**B*)  
*dBSRmat sparse matrix multiplication  $B = R * A * P$ , where small block matrices in P and R are identity matrices!*

### 9.35.1 Detailed Description

Linear algebraic operations for [dBSRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaSmallMat.c](#), and [BlaArray.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.35.2 Function Documentation

#### 9.35.2.1 fasp\_blas\_dbsr\_aAxpby()

```
void fasp_blas_dbsr_aAxpby (
    const REAL alpha,
    dBSRmat * A,
    REAL * x,
    const REAL beta,
    REAL * y )
```

Compute  $y := \alpha A * x + \beta y$ .

#### Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array x
<i>beta</i>	REAL factor beta
<i>y</i>	Pointer to the array y

#### Author

Zhiyang Zhou

#### Date

10/25/2010

Modified by Chunsheng Feng, Zheng Li on 06/29/2012

#### Note

Works for general nb (Xiaozhe)

Definition at line 67 of file BlaSpmvBSR.c.

#### 9.35.2.2 fasp\_blas\_dbsr\_aAxpby()

```
void fasp_blas_dbsr_aAxpby (
    const REAL alpha,
    const dBSRmat * A,
```

```
const REAL * x,  
REAL * y )
```

Compute  $y := \alpha * A * x + y$ .

## Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

## Author

Zhiyang Zhou

## Date

10/25/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

## Note

Works for general nb (Xiaozhe)

Definition at line 348 of file BlasSpmvBSR.c.

### 9.35.2.3 fasp\_blas\_dbsr\_aAxy\_agg()

```
void fasp_blas_dbsr_aAxy_agg (
    const REAL alpha,
    const dBSRmat * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := \alpha A * x + y$  where each small block matrix is an identity matrix.

## Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

## Author

Xiaozhe Hu

## Date

01/02/2014

## Note

Works for general nb (Xiaozhe)

Definition at line 624 of file BlasSpmvBSR.c.

### 9.35.2.4 fasp\_blas\_dbsr\_axm()

```
void fasp_blas_dbsr_axm (
    dBSRmat * A,
    const REAL alpha )
```

Multiply a sparse matrix A in BSR format by a scalar alpha.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> matrix A
<i>alpha</i>	REAL factor alpha

#### Author

Xiaozhe Hu

#### Date

05/26/2014

Definition at line 38 of file BlaSpmvBSR.c.

### 9.35.2.5 fasp\_blas\_dbsr\_mxm()

```
void fasp_blas_dbsr_mxm (
    const dBSRmat * A,
    const dBSRmat * B,
    dBSRmat * C )
```

Sparse matrix multiplication  $C=A*B$ .

#### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix A
<i>B</i>	Pointer to the <a href="#">dBSRmat</a> matrix B
<i>C</i>	Pointer to <a href="#">dBSRmat</a> matrix equal to $A*B$

#### Author

Xiaozhe Hu

#### Date

05/26/2014

#### Note

This fct will be replaced! – Xiaozhe

Definition at line 4646 of file BlaSpmvBSR.c.

### 9.35.2.6 fasp\_blas\_dbsr\_m xv()

```
void fasp_blas_dbsr_m xv (
    const dBSRmat * A,
```



```
const REAL * x,
REAL * y )
```

Compute  $y := A*x$ .

#### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

#### Author

Zhiyang Zhou

#### Date

10/25/2010

#### Note

Works for general nb (Xiaozhe)

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 910 of file BlasSpmvBSR.c.

#### 9.35.2.7 fasp\_blas\_dbsr\_mxv\_agg()

```
void fasp_blas_dbsr_mxv_agg (
    const dBSRmat * A,
    const REAL * x,
    REAL * y )
```

Compute  $y := A*x$ , where each small block matrices of  $A$  is an identity.

#### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

#### Author

Xiaozhe Hu

#### Date

01/02/2014

#### Note

Works for general nb (Xiaozhe)

Definition at line 2697 of file BlasSpmvBSR.c.

### 9.35.2.8 fasp\_blas\_dbsr\_rap()

```
void fasp_blas_dbsr_rap (
    const dBSRmat * R,
    const dBSRmat * A,
    const dBSRmat * P,
    dBSRmat * B )
```

**dBSRmat** sparse matrix multiplication  $B=R*A*P$

#### Parameters

<i>R</i>	Pointer to the <b>dBSRmat</b> matrix
<i>A</i>	Pointer to the <b>dBSRmat</b> matrix
<i>P</i>	Pointer to the <b>dBSRmat</b> matrix
<i>B</i>	Pointer to <b>dBSRmat</b> matrix equal to $R*A*P$ (output)

#### Author

Xiaozhe Hu, Chunsheng Feng, Zheng Li

#### Date

10/24/2012

#### Note

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 4961 of file BlaSpmvBSR.c.

### 9.35.2.9 fasp\_blas\_dbsr\_rap1()

```
void fasp_blas_dbsr_rap1 (
    const dBSRmat * R,
    const dBSRmat * A,
    const dBSRmat * P,
    dBSRmat * B )
```

**dBSRmat** sparse matrix multiplication  $B=R*A*P$

#### Parameters

<i>R</i>	Pointer to the <b>dBSRmat</b> matrix
<i>A</i>	Pointer to the <b>dBSRmat</b> matrix
<i>P</i>	Pointer to the <b>dBSRmat</b> matrix
<i>B</i>	Pointer to <b>dBSRmat</b> matrix equal to $R*A*P$ (output)

#### Author

Chunsheng Feng, Xiaoqiang Yue and Xiaozhe Hu

## Date

08/08/2011

## Note

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 4771 of file BlaSpmvBSR.c.

### 9.35.2.10 fasp\_blas\_dbsr\_rap\_agg()

```
void fasp_blas_dbsr_rap_agg (
    const dBSRmat * R,
    const dBSRmat * A,
    const dBSRmat * P,
    dBSRmat * B )
```

**dBSRmat** sparse matrix multiplication  $B=R*A*P$ , where small block matrices in  $P$  and  $R$  are identity matrices!

## Parameters

<i>R</i>	Pointer to the <b>dBSRmat</b> matrix
<i>A</i>	Pointer to the <b>dBSRmat</b> matrix
<i>P</i>	Pointer to the <b>dBSRmat</b> matrix
<i>B</i>	Pointer to <b>dBSRmat</b> matrix equal to $R*A*P$ (output)

## Author

Xiaozhe Hu

## Date

10/24/2012

Definition at line 5227 of file BlaSpmvBSR.c.

## 9.36 BlaSpmvCSR.c File Reference

Linear algebraic operations for **dCSRmat** matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- **SHORT** **fasp\_blas\_dcsr\_add** (const **dCSRmat** \*A, const **REAL** alpha, const **dCSRmat** \*B, const **REAL** beta, **dCSRmat** \*C)  
compute  $C = \alpha*A + \beta*B$  in CSR format
- void **fasp\_blas\_dcsr\_axm** (**dCSRmat** \*A, const **REAL** alpha)  
Multiply a sparse matrix A in CSR format by a scalar alpha.

- void `fasp_blas_dcsr_mxv` (const `dCSRmat` \*A, const `REAL` \*x, `REAL` \*y)  
*Matrix-vector multiplication  $y = A*x$ .*
- void `fasp_blas_dcsr_mxv_agg` (const `dCSRmat` \*A, const `REAL` \*x, `REAL` \*y)  
*Matrix-vector multiplication  $y = A*x$  (nonzeros of A = 1)*
- void `fasp_blas_dcsr_aAxy` (const `REAL` alpha, const `dCSRmat` \*A, const `REAL` \*x, `REAL` \*y)  
*Matrix-vector multiplication  $y = \alpha*A*x + y$ .*
- void `fasp_blas_dcsr_aAxy_agg` (const `REAL` alpha, const `dCSRmat` \*A, const `REAL` \*x, `REAL` \*y)  
*Matrix-vector multiplication  $y = \alpha*A*x + y$  (nonzeros of A = 1)*
- `REAL` `fasp_blas_dcsr_vmv` (const `dCSRmat` \*A, const `REAL` \*x, const `REAL` \*y)  
*vector-Matrix-vector multiplication  $\alpha = y'*A*x$*
- void `fasp_blas_dcsr_mxm` (const `dCSRmat` \*A, const `dCSRmat` \*B, `dCSRmat` \*C)  
*Sparse matrix multiplication  $C=A*B$ .*
- void `fasp_blas_dcsr_rap` (const `dCSRmat` \*R, const `dCSRmat` \*A, const `dCSRmat` \*P, `dCSRmat` \*RAP)  
*Triple sparse matrix multiplication  $B=R*A*P$ .*
- void `fasp_blas_dcsr_rap_agg` (const `dCSRmat` \*R, const `dCSRmat` \*A, const `dCSRmat` \*P, `dCSRmat` \*RAP)  
*Triple sparse matrix multiplication  $B=R*A*P$  (nonzeros of R, P = 1)*
- void `fasp_blas_dcsr_rap_agg1` (const `dCSRmat` \*R, const `dCSRmat` \*A, const `dCSRmat` \*P, `dCSRmat` \*B)  
*Triple sparse matrix multiplication  $B=R*A*P$  (nonzeros of R, P = 1)*
- void `fasp_blas_dcsr_ptap` (const `dCSRmat` \*Pt, const `dCSRmat` \*A, const `dCSRmat` \*P, `dCSRmat` \*Ac)  
*Triple sparse matrix multiplication  $B=P'*A*P$ .*
- `dCSRmat` `fasp_blas_dcsr_rap2` (`INT` \*ir, `INT` \*jr, `REAL` \*r, `INT` \*ia, `INT` \*ja, `REAL` \*a, `INT` \*ipt, `INT` \*jpt, `REAL` \*pt, `INT` n, `INT` nc, `INT` \*maxrpout, `INT` \*ipin, `INT` \*jpin)  
*Compute  $R*A*P$ .*
- void `fasp_blas_dcsr_rap4` (`dCSRmat` \*R, `dCSRmat` \*A, `dCSRmat` \*P, `dCSRmat` \*B, `INT` \*icor\_ysk)  
*Triple sparse matrix multiplication  $B=R*A*P$ .*

## Variables

- unsigned long `total_alloc_mem`
- unsigned long `total_alloc_count`

### 9.36.1 Detailed Description

Linear algebraic operations for `dCSRmat` matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaSparseCSR.c](#), [BlaSparseUtil.c](#), and [BlaArray.c](#)

Sparse functions usually contain three runs. The three runs are all the same but they serve different purpose.

Example: If you do `c=a+b`:

- first do a dry run to find the number of non-zeroes and form ic;
- allocate space (memory) for jc and form this one;
- if you only care about a "boolean" result of the addition, you stop here;
- you call another routine, which uses ic and jc to perform the addition.

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.36.2 Function Documentation

### 9.36.2.1 fasp\_blas\_dcsr\_aApy()

```
void fasp_blas_dcsr_aApy (
    const REAL alpha,
    const dCSRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = \alpha A x + y$ .

#### Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

#### Author

Chensong Zhang

#### Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/26/2012

Definition at line 489 of file BlasSpmvCSR.c.

### 9.36.2.2 fasp\_blas\_dcsr\_aApy\_agg()

```
void fasp_blas_dcsr_aApy_agg (
    const REAL alpha,
    const dCSRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = \alpha A x + y$  (nonzeros of  $A = 1$ )

#### Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

#### Author

Xiaozhe Hu

## Date

02/22/2011

Modified by Chunsheng Feng, Zheng Li on 08/29/2012  
 Definition at line 604 of file BlaspmvCSR.c.

**9.36.2.3 fasp\_blas\_dcsr\_add()**

```
SHORT fasp_blas_dcsr_add (
    const dCSRmat * A,
    const REAL alpha,
    const dCSRmat * B,
    const REAL beta,
    dCSRmat * C )
```

compute  $C = \alpha A + \beta B$  in CSR format

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix
<i>alpha</i>	REAL factor alpha
<i>B</i>	Pointer to <a href="#">dCSRmat</a> matrix
<i>beta</i>	REAL factor beta
<i>C</i>	Pointer to <a href="#">dCSRmat</a> matrix

## Returns

FASP\_SUCCESS if succeed, ERROR if not

## Author

Xiaozhe Hu

## Date

11/07/2009

Modified by Chunsheng Feng, Zheng Li on 06/29/2012  
 Definition at line 60 of file BlaspmvCSR.c.

**9.36.2.4 fasp\_blas\_dcsr\_axm()**

```
void fasp_blas_dcsr_axm (
    dCSRmat * A,
    const REAL alpha )
```

Multiply a sparse matrix A in CSR format by a scalar alpha.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix A
<i>alpha</i>	REAL factor alpha

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Zheng Li on 06/29/2012  
 Definition at line 212 of file BlasSpmvCSR.c.

**9.36.2.5 fasp\_blas\_dcsr\_mxm()**

```
void fasp_blas_dcsr_mxm (
    const dCSRmat * A,
    const dCSRmat * B,
    dCSRmat * C )
```

Sparse matrix multiplication  $C=A*B$ .

**Parameters**

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix A
<i>B</i>	Pointer to the <a href="#">dCSRmat</a> matrix B
<i>C</i>	Pointer to <a href="#">dCSRmat</a> matrix equal to $A*B$

**Author**

Xiaozhe Hu

**Date**

11/07/2009

**Warning**

This fct will be replaced! –Chensong

Definition at line 770 of file BlasSpmvCSR.c.

**9.36.2.6 fasp\_blas\_dcsr\_mxv()**

```
void fasp_blas_dcsr_mxv (
    const dCSRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = A*x$ .

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/26/2012  
 Definition at line 235 of file BlaspmvCSR.c.

**9.36.2.7 fasp\_blas\_dcsr\_mxv\_agg()**

```
void fasp_blas_dcsr_mxv_agg (
    const dCSRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = A*x$  (nonzeros of  $A = 1$ )

**Parameters**

<i>A</i>	Pointer to <code>dCSRmat</code> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

**Author**

Xiaozhe Hu

**Date**

02/22/2011

Modified by Chunsheng Feng, Zheng Li on 08/29/2012  
 Definition at line 432 of file BlaspmvCSR.c.

**9.36.2.8 fasp\_blas\_dcsr\_ptap()**

```
void fasp_blas_dcsr_ptap (
    const dCSRmat * Pt,
    const dCSRmat * A,
    const dCSRmat * P,
    dCSRmat * Ac )
```

Triple sparse matrix multiplication  $B=P'*A*P$ .

**Parameters**

<i>Pt</i>	Pointer to the restriction matrix
<i>A</i>	Pointer to the fine coefficient matrix
<i>P</i>	Pointer to the prolongation matrix
<i>Ac</i>	Pointer to the coarse coefficient matrix (output)



**Author**

Ludmil Zikatanov, Chensong Zhang

**Date**

05/10/2010

Modified by Chunsheng Feng, Zheng Li on 10/19/2012

**Note**

Driver to compute triple matrix product  $P^*A^*P$  using Itz CSR format. In Itz format:  $ia[0]=1$ ,  $ja[0]$  and  $a[0]$  are used as usual. When called from Fortran,  $ia[0]$ ,  $ja[0]$  and  $a[0]$  will be just  $ia(1), ja(1), a(1)$ . For the indices,  $ia\_ltz[k] = ia\_usual[k]+1$ ,  $ja\_ltz[k] = ja\_usual[k]+1$ ,  $a\_ltz[k] = a\_usual[k]$ .

Definition at line 1610 of file BlasSpmvCSR.c.

**9.36.2.9 fasp\_blas\_dcsr\_rap()**

```
void fasp_blas_dcsr_rap (
    const dCSRmat * R,
    const dCSRmat * A,
    const dCSRmat * P,
    dCSRmat * RAP )
```

Triple sparse matrix multiplication  $B=R*A*P$ .**Parameters**

<i>R</i>	Pointer to the <a href="#">dCSRmat</a> matrix R
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix A
<i>P</i>	Pointer to the <a href="#">dCSRmat</a> matrix P
<i>RAP</i>	Pointer to <a href="#">dCSRmat</a> matrix equal to $R*A*P$

**Author**

Xuehai Huang, Chensong Zhang

**Date**

05/10/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/26/2012

**Note**

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 878 of file BlasSpmvCSR.c.

**9.36.2.10 fasp\_blas\_dcsr\_rap2()**

```
dCSRmat fasp_blas_dcsr_rap2 (
    INT * ir,
    INT * jr,
```

```

REAL * r,
INT * ia,
INT * ja,
REAL * a,
INT * ipt,
INT * jpt,
REAL * pt,
INT n,
INT nc,
INT * maxrpout,
INT * ipin,
INT * jpin )

```

Compute  $R \cdot A \cdot P$ .

#### Author

Ludmil Zikatanov

#### Date

04/08/2010

#### Note

It uses [dCSRmat](#) only. The functions called from here are in `sparse_util.c`. Not used for the moment!

Definition at line 1710 of file `BlaSpmvCSR.c`.

### 9.36.2.11 fasp\_blas\_dcsr\_rap4()

```

void fasp_blas_dcsr_rap4 (
    dCSRmat * R,
    dCSRmat * A,
    dCSRmat * P,
    dCSRmat * B,
    INT * icor_ysk )

```

Triple sparse matrix multiplication  $B = R \cdot A \cdot P$ .

#### Parameters

<i>R</i>	pointer to the <a href="#">dCSRmat</a> matrix
<i>A</i>	pointer to the <a href="#">dCSRmat</a> matrix
<i>P</i>	pointer to the <a href="#">dCSRmat</a> matrix
<i>B</i>	pointer to <a href="#">dCSRmat</a> matrix equal to $R \cdot A \cdot P$
<i>icor_ysk</i>	pointer to the array

#### Author

Feng Chunsheng, Yue Xiaoqiang

#### Date

08/02/2011

**Note**

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 1808 of file BlasSpmvCSR.c.

**9.36.2.12 fasp\_blas\_dcsr\_rap\_agg()**

```
void fasp_blas_dcsr_rap_agg (
    const dCSRmat * R,
    const dCSRmat * A,
    const dCSRmat * P,
    dCSRmat * RAP )
```

Triple sparse matrix multiplication  $B=R*A*P$  (nonzeros of R,  $P = 1$ )

**Parameters**

<i>R</i>	Pointer to the <a href="#">dCSRmat</a> matrix R
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix A
<i>P</i>	Pointer to the <a href="#">dCSRmat</a> matrix P
<i>RAP</i>	Pointer to <a href="#">dCSRmat</a> matrix equal to $R*A*P$

**Author**

Xiaozhe Hu

**Date**

05/10/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/26/2012

Definition at line 1158 of file BlasSpmvCSR.c.

**9.36.2.13 fasp\_blas\_dcsr\_rap\_agg1()**

```
void fasp_blas_dcsr_rap_agg1 (
    const dCSRmat * R,
    const dCSRmat * A,
    const dCSRmat * P,
    dCSRmat * B )
```

Triple sparse matrix multiplication  $B=R*A*P$  (nonzeros of R,  $P = 1$ )

**Parameters**

<i>R</i>	Pointer to the <a href="#">dCSRmat</a> matrix R
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix A
<i>P</i>	Pointer to the <a href="#">dCSRmat</a> matrix P
<i>B</i>	Pointer to <a href="#">dCSRmat</a> matrix equal to $R*A*P$

**Author**

Xiaozhe Hu

**Date**

02/21/2011

**Note**

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 1424 of file BlaSpmvCSR.c.

**9.36.2.14 fasp\_blas\_dcsr\_vmv()**

```
REAL fasp_blas_dcsr_vmv (
    const dCSRmat * A,
    const REAL * x,
    const REAL * y )
```

vector-Matrix-vector multiplication  $\alpha = y'A*x$

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

**Author**

Chensong Zhang

**Date**

07/01/2009

Definition at line 715 of file BlaSpmvCSR.c.

**9.37 BlaSpmvCSRL.c File Reference**

Linear algebraic operations for [dCSRLmat](#) matrices.

```
#include "fasp.h"
```

**Functions**

- void [fasp\\_blas\\_dcsrl\\_mxv](#) (const [dCSRLmat](#) \*A, const [REAL](#) \*x, [REAL](#) \*y)  
*Compute  $y = A*x$  for a sparse matrix in CSRL format.*

**9.37.1 Detailed Description**

Linear algebraic operations for [dCSRLmat](#) matrices.

**Note**

This file contains Level-1 (Bla) functions.

Reference: John Mellor-Crummey and John Garvin Optimizaing sparse matrix vector product computations using unroll and jam, Tech Report Rice Univ, Aug 2002.

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

**9.37.2 Function Documentation****9.37.2.1 fasp\_blas\_dcsr\_lmxv()**

```
void fasp_blas_dcsr_lmxv (
    const dCSRmat * A,
    const REAL * x,
    REAL * y )
```

Compute  $y = A*x$  for a sparse matrix in CSR format.

**Parameters**

<i>A</i>	Pointer to dCSRmat matrix A
<i>x</i>	Pointer to REAL array of vector x
<i>y</i>	Pointer to REAL array of vector y

**Author**

Zhiyang Zhou, Chensong Zhang

**Date**

2011/01/07

Definition at line 36 of file BlasPmvCSR.c.

**9.38 BlasPmvSTR.c File Reference**

Linear algebraic operations for dSTRmat matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void **fasp\_blas\_dstr\_aAxy** (const REAL alpha, const dSTRmat \*A, const REAL \*x, REAL \*y)  
*Matrix-vector multiplication  $y = \alpha * A * x + y$ .*
- void **fasp\_blas\_dstr\_mxy** (const dSTRmat \*A, const REAL \*x, REAL \*y)  
*Matrix-vector multiplication  $y = A * x$ .*
- INT **fasp\_blas\_dstr\_diagscale** (const dSTRmat \*A, dSTRmat \*B)  
 *$B = D^{-1}A$ .*

### 9.38.1 Detailed Description

Linear algebraic operations for [dSTRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaSmallMatInv.c](#), [BlaSmallMat.c](#), and [BlaSparseSTR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.38.2 Function Documentation

#### 9.38.2.1 fasp\_blas\_dstr\_aApy()

```
void fasp_blas_dstr_aApy (
    const REAL alpha,
    const dSTRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = \alpha A x + y$ .

#### Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to <a href="#">dSTRmat</a> matrix
<i>x</i>	Pointer to REAL array
<i>y</i>	Pointer to REAL array

#### Author

Zhiyang Zhou, Xiaozhe Hu, Shiquan Zhang

#### Date

2010/10/15

Definition at line 61 of file [BlaSpmvSTR.c](#).

#### 9.38.2.2 fasp\_blas\_dstr\_diagscale()

```
INT fasp_blas_dstr_diagscale (
    const dSTRmat * A,
    dSTRmat * B )
```

$B = D^{-1}A$ .

#### Parameters

<i>A</i>	Pointer to a ' <a href="#">dSTRmat</a> ' type matrix A
<i>B</i>	Pointer to a ' <a href="#">dSTRmat</a> ' type matrix B

## Author

Shiquan Zhang

## Date

2010/10/15

Modified by Chunsheng Feng, Zheng Li on 08/30/2012  
 Definition at line 155 of file BlaspmvSTR.c.

## 9.38.2.3 fasp\_blas\_dstr\_mvx()

```
void fasp_blas_dstr_mvx (
    const dSTRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = A*x$ .

## Parameters

<i>A</i>	Pointer to <code>dSTRmat</code> matrix
<i>x</i>	Pointer to REAL array
<i>y</i>	Pointer to REAL array

## Author

Chensong Zhang

## Date

04/27/2013

Definition at line 131 of file BlaspmvSTR.c.

## 9.39 BlaVector.c File Reference

BLAS1 operations for vectors.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- void `fasp_blas_dvec_axpy` (const `REAL` *a*, const `dvector` \**x*, `dvector` \**y*)  
 $y = a*x + y$
- void `fasp_blas_dvec_axpyz` (const `REAL` *a*, const `dvector` \**x*, const `dvector` \**y*, `dvector` \**z*)  
 $z = a*x + y$ , *z* is a third vector (*z* is cleared)
- `REAL` `fasp_blas_dvec_norm1` (const `dvector` \**x*)  
 $L_1$  norm of `dvector` *x*.
- `REAL` `fasp_blas_dvec_norm2` (const `dvector` \**x*)

*L2 norm of dvector x.*

- [REAL fasp\\_blas\\_dvec\\_norminf](#) (const [dvector](#) \*x)

*Linf norm of dvector x.*

- [REAL fasp\\_blas\\_dvec\\_dotprod](#) (const [dvector](#) \*x, const [dvector](#) \*y)

*Inner product of two vectors (x,y)*

- [REAL fasp\\_blas\\_dvec\\_relerr](#) (const [dvector](#) \*x, const [dvector](#) \*y)

*Relative difference between two dvector x and y.*

### 9.39.1 Detailed Description

BLAS1 operations for vectors.

#### Note

This file contains Level-1 (Bla) functions. It requires: [AuxMessage.c](#), [AuxThreads.c](#), and [BlaArray.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.39.2 Function Documentation

#### 9.39.2.1 fasp\_blas\_dvec\_axpy()

```
void fasp_blas_dvec_axpy (
    const REAL a,
    const dvector * x,
    dvector * y )
```

$y = a \cdot x + y$

#### Parameters

<i>a</i>	REAL factor a
<i>x</i>	Pointer to dvector x
<i>y</i>	Pointer to dvector y

#### Author

Chensong Zhang

#### Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 41 of file BlaVector.c.

#### 9.39.2.2 fasp\_blas\_dvec\_axpyz()

```
void fasp_blas_dvec_axpyz (
    const REAL a,
    const dvector * x,
```



```
const dvector * y,  
dvector * z )
```

$z = a * x + y$ ,  $z$  is a third vector ( $z$  is cleared)

#### Parameters

$a$	REAL factor $a$
$x$	Pointer to dvector $x$
$y$	Pointer to dvector $y$
$z$	Pointer to dvector $z$

#### Author

Chensong Zhang

#### Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 96 of file BlaVector.c.

### 9.39.2.3 fasp\_blas\_dvec\_dotprod()

```
REAL fasp_blas_dvec_dotprod (  
    const dvector * x,  
    const dvector * y )
```

Inner product of two vectors ( $x, y$ )

#### Parameters

$x$	Pointer to dvector $x$
$y$	Pointer to dvector $y$

#### Returns

Inner product

#### Author

Chensong Zhang

#### Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 236 of file BlaVector.c.

### 9.39.2.4 fasp\_blas\_dvec\_norm1()

```
REAL fasp_blas_dvec_norm1 (  
    const dvector * x )
```

L1 norm of dvector  $x$ .

**Parameters**

$x$	Pointer to dvector $x$
-----	------------------------

**Returns**

L1 norm of  $x$

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012  
Definition at line 130 of file BlaVector.c.

**9.39.2.5 fasp\_blas\_dvec\_norm2()**

```
REAL fasp_blas_dvec_norm2 (  
    const dvector *  $x$  )
```

L2 norm of dvector  $x$ .

**Parameters**

$x$	Pointer to dvector $x$
-----	------------------------

**Returns**

L2 norm of  $x$

**Author**

Chensong Zhang

**Date**

07/01/2009

Definition at line 170 of file BlaVector.c.

**9.39.2.6 fasp\_blas\_dvec\_norminf()**

```
REAL fasp_blas_dvec_norminf (  
    const dvector *  $x$  )
```

Linf norm of dvector  $x$ .

**Parameters**

$x$	Pointer to dvector $x$
-----	------------------------

**Returns**

$L_{\infty}$  norm of  $x$

**Author**

Chensong Zhang

**Date**

07/01/2009

Definition at line 208 of file BlaVector.c.

**9.39.2.7 fasp\_blas\_dvec\_relerr()**

```
REAL fasp_blas_dvec_relerr (
    const dvector * x,
    const dvector * y )
```

Relative difference between two dvector  $x$  and  $y$ .

**Parameters**

$x$	Pointer to dvector $x$
$y$	Pointer to dvector $y$

**Returns**

Relative difference  $\|x-y\|/\|x\|$

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012

Definition at line 278 of file BlaVector.c.

**9.40 doxygen.h File Reference**

Main page for Doygen documentation.

**9.40.1 Detailed Description**

Main page for Doygen documentation.

Copyright (C) 2010–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Less Public License 3.0 or later.

**9.41 fasp.h File Reference**

Main header file for the FASP project.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fasp_const.h"
```

## Data Structures

- struct [ddenmat](#)  
*Dense matrix of REAL type.*
- struct [idenmat](#)  
*Dense matrix of INT type.*
- struct [dCSRmat](#)  
*Sparse matrix of REAL type in CSR format.*
- struct [iCSRmat](#)  
*Sparse matrix of INT type in CSR format.*
- struct [dCOOmat](#)  
*Sparse matrix of REAL type in COO (IJ) format.*
- struct [iCOOmat](#)  
*Sparse matrix of INT type in COO (IJ) format.*
- struct [dCSRLmat](#)  
*Sparse matrix of REAL type in CSRL format.*
- struct [dSTRmat](#)  
*Structure matrix of REAL type.*
- struct [dvector](#)  
*Vector with  $n$  entries of REAL type.*
- struct [ivector](#)  
*Vector with  $n$  entries of INT type.*
- struct [ITS\\_param](#)  
*Parameters for iterative solvers.*
- struct [ILU\\_param](#)  
*Parameters for ILU.*
- struct [SWZ\\_param](#)  
*Parameters for Schwarz method.*
- struct [AMG\\_param](#)  
*Parameters for AMG methods.*
- struct [Mumps\\_data](#)  
*Data for MUMPS interface.*
- struct [Pardiso\\_data](#)  
*Data for Intel MKL PARDISO interface.*
- struct [ILU\\_data](#)  
*Data for ILU setup.*
- struct [SWZ\\_data](#)  
*Data for Schwarz methods.*
- struct [AMG\\_data](#)  
*Data for AMG methods.*
- struct [precond\\_data](#)  
*Data for preconditioners.*

- struct [precond\\_data\\_str](#)  
*Data for preconditioners in dSTRmat format.*
- struct [precond\\_diag\\_str](#)  
*Data for diagonal preconditioners in dSTRmat format.*
- struct [precond](#)  
*Preconditioner data and action.*
- struct [mxv\\_matfree](#)  
*Matrix-vector multiplication, replace the actual matrix.*
- struct [input\\_param](#)  
*Input parameters.*

## Macros

- `#define __FASP_HEADER__`
- `#define FASP_VERSION 2.0`  
*FASP base version information.*
- `#define DLMALLOC OFF`  
*For external software package support.*
- `#define NEDMALLOC OFF`
- `#define RS_C1 ON`  
*Flags for internal uses.*
- `#define DIAGONAL_PREF OFF`
- `#define SHORT short`  
*FASP integer and floating point numbers.*
- `#define INT int`
- `#define LONG long`
- `#define LONGLONG long long`
- `#define REAL double`
- `#define STRLEN 256`
- `#define MAX(a, b) (((a)>(b))?(a):(b))`  
*Definition of max, min, abs.*
- `#define MIN(a, b) (((a)<(b))?(a):(b))`
- `#define ABS(a) (((a)>=0.0)?(a):-(a))`
- `#define GT(a, b) (((a)>(b))?(TRUE):(FALSE))`  
*Definition of >, >=, <, <=, and isnan.*
- `#define GE(a, b) (((a)>=(b))?(TRUE):(FALSE))`
- `#define LS(a, b) (((a)<(b))?(TRUE):(FALSE))`
- `#define LE(a, b) (((a)<=(b))?(TRUE):(FALSE))`
- `#define ISNAN(a) (((a)!=a)?(TRUE):(FALSE))`
- `#define PUT_INT(A) printf("### DEBUG: %s = %d\n", #A, (A))`  
*Definition of print command in DEBUG mode.*
- `#define PUT_REAL(A) printf("### DEBUG: %s = %e\n", #A, (A))`

## Typedefs

- typedef struct [ddenmat](#) ddenmat
- typedef struct [idenmat](#) idenmat
- typedef struct [dCSRmat](#) dCSRmat
- typedef struct [iCSRmat](#) iCSRmat
- typedef struct [dCOOmat](#) dCOOmat
- typedef struct [iCOOmat](#) iCOOmat
- typedef struct [dCSRLmat](#) dCSRLmat
- typedef struct [dSTRmat](#) dSTRmat
- typedef struct [dvector](#) dvector
- typedef struct [ivector](#) ivector

### 9.41.1 Detailed Description

Main header file for the FASP project.

#### Note

This header file contains general constants and data structures of FASP. It contains macros and data structure definitions; should not include function declarations here.

---

Copyright (C) 2008–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.41.2 Macro Definition Documentation

#### 9.41.2.1 `__FASP_HEADER__`

```
#define __FASP_HEADER__
```

indicate [fasp.h](#) has been included before  
Definition at line 31 of file fasp.h.

#### 9.41.2.2 `ABS`

```
#define ABS( a ) ((a)>=0.0)?(a):-(a)
```

absolute value of a  
Definition at line 73 of file fasp.h.

#### 9.41.2.3 `DIAGONAL_PREF`

```
#define DIAGONAL_PREF OFF
```

order each row such that diagonal appears first  
Definition at line 56 of file fasp.h.

#### 9.41.2.4 DLMALLOC

```
#define DLMALLOC OFF
```

For external software package support.  
use dmalloc instead of standard malloc  
Definition at line 45 of file fasp.h.

#### 9.41.2.5 FASP\_VERSION

```
#define FASP_VERSION 2.0
```

FASP base version information.  
fapsolver version  
Definition at line 40 of file fasp.h.

#### 9.41.2.6 GE

```
#define GE(  
    a,  
    b ) ((a) >= (b)) ? (TRUE) : (FALSE)
```

is a >= b?

Definition at line 79 of file fasp.h.

#### 9.41.2.7 GT

```
#define GT(  
    a,  
    b ) ((a) > (b)) ? (TRUE) : (FALSE)
```

Definition of >, >=, <, <=, and isnan.

is a > b?

Definition at line 78 of file fasp.h.

#### 9.41.2.8 INT

```
#define INT int
```

signed integer types: signed, long enough

Definition at line 62 of file fasp.h.

#### 9.41.2.9 ISNAN

```
#define ISNAN(  
    a ) ((a) != (a)) ? (TRUE) : (FALSE)
```

is a == NAN?

Definition at line 82 of file fasp.h.

#### 9.41.2.10 LE

```
#define LE(  
    a,  
    b ) ((a) <= (b)) ? (TRUE) : (FALSE)
```

is  $a \leq b$ ?

Definition at line 81 of file fasp.h.

#### 9.41.2.11 LONG

```
#define LONG long
```

long integer type

Definition at line 63 of file fasp.h.

#### 9.41.2.12 LONGLONG

```
#define LONGLONG long long
```

long long integer type

Definition at line 64 of file fasp.h.

#### 9.41.2.13 LS

```
#define LS(
```

```
    a,
```

```
    b ) ((a)<(b))?(TRUE):(FALSE))
```

is  $a < b$ ?

Definition at line 80 of file fasp.h.

#### 9.41.2.14 MAX

```
#define MAX(
```

```
    a,
```

```
    b ) ((a)>(b))?(a):(b))
```

Definition of max, min, abs.

bigger one in a and b

Definition at line 71 of file fasp.h.

#### 9.41.2.15 MIN

```
#define MIN(
```

```
    a,
```

```
    b ) ((a)<(b))?(a):(b))
```

smaller one in a and b

Definition at line 72 of file fasp.h.

#### 9.41.2.16 NEDMALLOC

```
#define NEDMALLOC OFF
```

use nedmalloc instead of standard malloc

Definition at line 46 of file fasp.h.



#### 9.41.2.17 PUT\_INT

```
#define PUT_INT(  
    A ) printf("### DEBUG: %s = %d\n", #A, (A))
```

Definition of print command in DEBUG mode.

print integer

Definition at line 87 of file fasp.h.

#### 9.41.2.18 PUT\_REAL

```
#define PUT_REAL(  
    A ) printf("### DEBUG: %s = %e\n", #A, (A))
```

print real num

Definition at line 88 of file fasp.h.

#### 9.41.2.19 REAL

```
#define REAL double
```

float type

Definition at line 65 of file fasp.h.

#### 9.41.2.20 RS\_C1

```
#define RS_C1 ON
```

Flags for internal uses.

##### Warning

Change the following marcos with caution! CF splitting of RS: check C1 Criterion

Definition at line 54 of file fasp.h.

#### 9.41.2.21 SHORT

```
#define SHORT short
```

FASP integer and floating point numbers.

short integer type

Definition at line 61 of file fasp.h.

#### 9.41.2.22 STRLEN

```
#define STRLEN 256
```

length of strings

Definition at line 66 of file fasp.h.

### 9.41.3 Typedef Documentation

#### 9.41.3.1 dCOOmat

```
typedef struct dCOOmat dCOOmat
```

Sparse matrix of REAL type in COO format

#### 9.41.3.2 dCSRLmat

```
typedef struct dCSRLmat dCSRLmat
```

Sparse matrix of REAL type in CSRL format

#### 9.41.3.3 dCSRmat

```
typedef struct dCSRmat dCSRmat
```

Sparse matrix of REAL type in CSR format

#### 9.41.3.4 ddenmat

```
typedef struct ddenmat ddenmat
```

Dense matrix of REAL type

#### 9.41.3.5 dSTRmat

```
typedef struct dSTRmat dSTRmat
```

Structured matrix of REAL type

#### 9.41.3.6 dvector

```
typedef struct dvector dvector
```

Vector of REAL type

#### 9.41.3.7 iCOOmat

```
typedef struct iCOOmat iCOOmat
```

Sparse matrix of INT type in COO format

#### 9.41.3.8 iCSRmat

```
typedef struct iCSRmat iCSRmat
```

Sparse matrix of INT type in CSR format

#### 9.41.3.9 idenmat

```
typedef struct idenmat idenmat
```

Dense matrix of INT type

#### 9.41.3.10 ivector

```
typedef struct ivector ivector
```

Vector of INT type

## 9.42 fasp\_block.h File Reference

Header file for FASP block matrices.

```
#include "fasp.h"
```

## Data Structures

- struct [dBSRmat](#)  
*Block sparse row storage matrix of REAL type.*
- struct [dBLCmat](#)  
*Block REAL CSR matrix format.*
- struct [iBLCmat](#)  
*Block INT CSR matrix format.*
- struct [block\\_dvector](#)  
*Block REAL vector structure.*
- struct [block\\_ivector](#)  
*Block INT vector structure.*
- struct [AMG\\_data\\_bsr](#)  
*Data for multigrid levels in [dBSRmat](#) format.*
- struct [precond\\_diag\\_bsr](#)  
*Data for diagonal preconditioners in [dBSRmat](#) format.*
- struct [precond\\_data\\_bsr](#)  
*Data for preconditioners in [dBSRmat](#) format.*
- struct [precond\\_data\\_blc](#)  
*Data for block preconditioners in [dBLCmat](#) format.*
- struct [precond\\_data\\_sweeping](#)  
*Data for sweeping preconditioner.*

## Macros

- `#define \_\_FASPBLOCK\_HEADER\_\_`

## Typedefs

- typedef struct [dBSRmat](#) [dBSRmat](#)
- typedef struct [dBLCmat](#) [dBLCmat](#)
- typedef struct [iBLCmat](#) [iBLCmat](#)
- typedef struct [block\\_dvector](#) [block\\_dvector](#)
- typedef struct [block\\_ivector](#) [block\\_ivector](#)

### 9.42.1 Detailed Description

Header file for FASP block matrices.

#### Note

This header file contains definitions of block matrices, including grid-major type and variable-major type. In this header, we only define macros and data structures, not function declarations.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.42.2 Macro Definition Documentation

### 9.42.2.1 `__FASPBLOCK_HEADER__`

`#define __FASPBLOCK_HEADER__`  
 indicate [fasp\\_block.h](#) has been included before  
 Definition at line 18 of file `fasp_block.h`.

## 9.42.3 Typedef Documentation

### 9.42.3.1 `block_dvector`

`typedef struct block_dvector block_dvector`  
 Vector of REAL type in Block format

### 9.42.3.2 `block_ivec`

`typedef struct block_ivec block_ivec`  
 Vector of INT type in Block format

### 9.42.3.3 `dBLCmat`

`typedef struct dBLCmat dBLCmat`  
 Matrix of REAL type in Block CSR format

### 9.42.3.4 `dB SRmat`

`typedef struct dB SRmat dB SRmat`  
 Matrix of REAL type in BSR format

### 9.42.3.5 `iBLCmat`

`typedef struct iBLCmat iBLCmat`  
 Matrix of INT type in Block CSR format

## 9.43 `fasp_const.h` File Reference

Definition of FASP constants, including messages, solver types, etc.

### Macros

- `#define FASP_SUCCESS 0`  
*Definition of return status and error messages.*
- `#define ERROR_READ_FILE -1`
- `#define ERROR_OPEN_FILE -10`
- `#define ERROR_WRONG_FILE -11`
- `#define ERROR_INPUT_PAR -13`
- `#define ERROR_REGRESS -14`
- `#define ERROR_MAT_SIZE -15`
- `#define ERROR_NUM_BLOCKS -18`
- `#define ERROR_MISC -19`
- `#define ERROR_ALLOC_MEM -20`
- `#define ERROR_DATA_STRUCTURE -21`

- #define [ERROR\\_DATA\\_ZERODIAG](#) -22
- #define [ERROR\\_DUMMY\\_VAR](#) -23
- #define [ERROR\\_AMG\\_INTERP\\_TYPE](#) -30
- #define [ERROR\\_AMG\\_SMOOTH\\_TYPE](#) -31
- #define [ERROR\\_AMG\\_COARSE\\_TYPE](#) -32
- #define [ERROR\\_AMG\\_COARSEING](#) -33
- #define [ERROR\\_AMG\\_SETUP](#) -39
- #define [ERROR\\_SOLVER\\_TYPE](#) -40
- #define [ERROR\\_SOLVER\\_PRECTYPE](#) -41
- #define [ERROR\\_SOLVER\\_STAG](#) -42
- #define [ERROR\\_SOLVER\\_SOLSTAG](#) -43
- #define [ERROR\\_SOLVER\\_TOLSMALL](#) -44
- #define [ERROR\\_SOLVER\\_ILUSETUP](#) -45
- #define [ERROR\\_SOLVER\\_MISC](#) -46
- #define [ERROR\\_SOLVER\\_MAXIT](#) -48
- #define [ERROR\\_SOLVER\\_EXIT](#) -49
- #define [ERROR\\_QUAD\\_TYPE](#) -60
- #define [ERROR\\_QUAD\\_DIM](#) -61
- #define [ERROR\\_LIC\\_TYPE](#) -80
- #define [ERROR\\_UNKNOWN](#) -99
- #define [TRUE](#) 1
- *Definition of logic type.*
- #define [FALSE](#) 0
- #define [ON](#) 1
- *Definition of switch.*
- #define [OFF](#) 0
- #define [PRINT\\_NONE](#) 0
- *Print level for all subroutines – not including DEBUG output.*
- #define [PRINT\\_MIN](#) 1
- #define [PRINT\\_SOME](#) 2
- #define [PRINT\\_MORE](#) 4
- #define [PRINT\\_MOST](#) 8
- #define [PRINT\\_ALL](#) 10
- #define [MAT\\_FREE](#) 0
- *Definition of matrix format.*
- #define [MAT\\_CSR](#) 1
- #define [MAT\\_BSR](#) 2
- #define [MAT\\_STR](#) 3
- #define [MAT\\_CSRL](#) 6
- #define [MAT\\_SymCSR](#) 7
- #define [MAT\\_BLC](#) 8
- #define [MAT\\_bCSR](#) 11
- #define [MAT\\_bBSR](#) 12
- #define [MAT\\_bSTR](#) 13
- #define [SOLVER\\_DEFAULT](#) 0
- *Definition of solver types for iterative methods.*
- #define [SOLVER\\_CG](#) 1
- #define [SOLVER\\_BiCGstab](#) 2
- #define [SOLVER\\_MinRes](#) 3
- #define [SOLVER\\_GMRES](#) 4

- #define SOLVER\_VGMRES 5
- #define SOLVER\_VFGMRES 6
- #define SOLVER\_GCG 7
- #define SOLVER\_GCR 8
- #define SOLVER\_SCG 11
- #define SOLVER\_SBiGstab 12
- #define SOLVER\_SMinRes 13
- #define SOLVER\_SGMRES 14
- #define SOLVER\_SVGMRES 15
- #define SOLVER\_SVFGMRES 16
- #define SOLVER\_SGCG 17
- #define SOLVER\_AMG 21
- #define SOLVER\_FMG 22
- #define SOLVER\_SUPERLU 31
- #define SOLVER\_UMFPACK 32
- #define SOLVER\_MUMPS 33
- #define SOLVER\_PARDISO 34
- #define STOP\_REL\_RES 1

*Definition of iterative solver stopping criteria types.*

- #define STOP\_REL\_PRECRES 2
- #define STOP\_MOD\_REL\_RES 3
- #define PREC\_NULL 0

*Definition of preconditioner type for iterative methods.*

- #define PREC\_DIAG 1
- #define PREC\_AMG 2
- #define PREC\_FMG 3
- #define PREC\_ILU 4
- #define PREC\_SCHWARZ 5
- #define ILUk 1

*Type of ILU methods.*

- #define ILUt 2
- #define ILUtp 3
- #define SCHWARZ\_FORWARD 1

*Type of Schwarz smoother.*

- #define SCHWARZ\_BACKWARD 2
- #define SCHWARZ\_SYMMETRIC 3
- #define CLASSIC\_AMG 1

*Definition of AMG types.*

- #define SA\_AMG 2
- #define UA\_AMG 3
- #define PAIRWISE 1

*Definition of aggregation types.*

- #define VMB 2
- #define NPAIR 3
- #define SPAIR 4
- #define V\_CYCLE 1

*Definition of cycle types.*

- #define W\_CYCLE 2
- #define AMLI\_CYCLE 3
- #define NL\_AMLI\_CYCLE 4

- #define [VW\\_CYCLE](#) 12
- #define [WV\\_CYCLE](#) 21
- #define [SMOOTHER\\_JACOBI](#) 1

*Definition of standard smoother types.*

- #define [SMOOTHER\\_GS](#) 2
- #define [SMOOTHER\\_SGS](#) 3
- #define [SMOOTHER\\_CG](#) 4
- #define [SMOOTHER\\_SOR](#) 5
- #define [SMOOTHER\\_SSOR](#) 6
- #define [SMOOTHER\\_GSOR](#) 7
- #define [SMOOTHER\\_SGSOR](#) 8
- #define [SMOOTHER\\_POLY](#) 9
- #define [SMOOTHER\\_L1DIAG](#) 10
- #define [SMOOTHER\\_BLKOIL](#) 11

*Definition of specialized smoother types.*

- #define [SMOOTHER\\_SPETEN](#) 19
- #define [COARSE\\_RS](#) 1

*Definition of coarsening types.*

- #define [COARSE\\_RSP](#) 2
- #define [COARSE\\_CR](#) 3
- #define [COARSE\\_AC](#) 4
- #define [COARSE\\_MIS](#) 5
- #define [INTERP\\_DIR](#) 1

*Definition of interpolation types.*

- #define [INTERP\\_STD](#) 2
- #define [INTERP\\_ENG](#) 3
- #define [INTERP\\_EXT](#) 6
- #define [G0PT](#) -5

*Type of vertices (DOFs) for coarsening.*

- #define [UNPT](#) -1
- #define [FGPT](#) 0
- #define [CGPT](#) 1
- #define [ISPT](#) 2
- #define [NO\\_ORDER](#) 0

*Definition of smoothing order.*

- #define [CF\\_ORDER](#) 1
- #define [USERDEFINED](#) 0

*Type of ordering for smoothers.*

- #define [CPFIRST](#) 1
- #define [FPFIRST](#) -1
- #define [ASCEND](#) 12
- #define [DESCEND](#) 21
- #define [BIGREAL](#) 1e+20

*Some global constants.*

- #define [SMALLREAL](#) 1e-20
- #define [SMALLREAL2](#) 1e-40
- #define [MAX\\_REFINE\\_LVL](#) 20
- #define [MAX\\_AMG\\_LVL](#) 20
- #define [MIN\\_CDOF](#) 20

- `#define MIN_CRATE 0.9`
- `#define MAX_CRATE 20.0`
- `#define MAX_RESTART 20`
- `#define MAX_STAG 20`
- `#define STAG_RATIO 1e-4`
- `#define OPENMP_HOLDS 2000`

### 9.43.1 Detailed Description

Definition of FASP constants, including messages, solver types, etc.  
Copyright (C) 2009–2020 by the FASP team. All rights reserved.

---

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

Warning

This is for internal use only. Do NOT change!

### 9.43.2 Macro Definition Documentation

#### 9.43.2.1 AMLI\_CYCLE

```
#define AMLI_CYCLE 3
```

AMLI-cycle

Definition at line 179 of file fasp\_const.h.

#### 9.43.2.2 ASCEND

```
#define ASCEND 12
```

Ascending order

Definition at line 242 of file fasp\_const.h.

#### 9.43.2.3 BIGREAL

```
#define BIGREAL 1e+20
```

Some global constants.

A large real number

Definition at line 248 of file fasp\_const.h.

#### 9.43.2.4 CF\_ORDER

```
#define CF_ORDER 1
```

C/F order smoothing

Definition at line 234 of file fasp\_const.h.



#### 9.43.2.5 CGPT

```
#define CGPT 1
```

Coarse grid points  
Definition at line 227 of file fasp\_const.h.

#### 9.43.2.6 CLASSIC\_AMG

```
#define CLASSIC_AMG 1
```

Definition of AMG types.  
classic AMG  
Definition at line 162 of file fasp\_const.h.

#### 9.43.2.7 COARSE\_AC

```
#define COARSE_AC 4
```

Aggressive coarsening  
Definition at line 210 of file fasp\_const.h.

#### 9.43.2.8 COARSE\_CR

```
#define COARSE_CR 3
```

Compatible relaxation  
Definition at line 209 of file fasp\_const.h.

#### 9.43.2.9 COARSE\_MIS

```
#define COARSE_MIS 5
```

Aggressive coarsening based on MIS  
Definition at line 211 of file fasp\_const.h.

#### 9.43.2.10 COARSE\_RS

```
#define COARSE_RS 1
```

Definition of coarsening types.  
Classical  
Definition at line 207 of file fasp\_const.h.

#### 9.43.2.11 COARSE\_RSP

```
#define COARSE_RSP 2
```

Classical, with positive offdiags  
Definition at line 208 of file fasp\_const.h.

#### 9.43.2.12 CPFIRST

```
#define CPFIRST 1
```

C-points first order  
Definition at line 240 of file fasp\_const.h.

#### 9.43.2.13 DESCEND

```
#define DESCEND 21
```

Descending order

Definition at line 243 of file fasp\_const.h.

#### 9.43.2.14 ERROR\_ALLOC\_MEM

```
#define ERROR_ALLOC_MEM -20
```

fail to allocate memory

Definition at line 30 of file fasp\_const.h.

#### 9.43.2.15 ERROR\_AMG\_COARSE\_TYPE

```
#define ERROR_AMG_COARSE_TYPE -32
```

unknown coarsening type

Definition at line 37 of file fasp\_const.h.

#### 9.43.2.16 ERROR\_AMG\_COARSEING

```
#define ERROR_AMG_COARSEING -33
```

coarsening step failed to complete

Definition at line 38 of file fasp\_const.h.

#### 9.43.2.17 ERROR\_AMG\_INTERP\_TYPE

```
#define ERROR_AMG_INTERP_TYPE -30
```

unknown interpolation type

Definition at line 35 of file fasp\_const.h.

#### 9.43.2.18 ERROR\_AMG\_SETUP

```
#define ERROR_AMG_SETUP -39
```

AMG setup failed to complete

Definition at line 39 of file fasp\_const.h.

#### 9.43.2.19 ERROR\_AMG\_SMOOTH\_TYPE

```
#define ERROR_AMG_SMOOTH_TYPE -31
```

unknown smoother type

Definition at line 36 of file fasp\_const.h.

#### 9.43.2.20 ERROR\_DATA\_STRUCTURE

```
#define ERROR_DATA_STRUCTURE -21
```

problem with data structures

Definition at line 31 of file fasp\_const.h.

#### 9.43.2.21 ERROR\_DATA\_ZERODIAG

```
#define ERROR_DATA_ZERODIAG -22
```

matrix has zero diagonal entries  
Definition at line 32 of file fasp\_const.h.

#### 9.43.2.22 ERROR\_DUMMY\_VAR

```
#define ERROR_DUMMY_VAR -23
```

unexpected input data  
Definition at line 33 of file fasp\_const.h.

#### 9.43.2.23 ERROR\_INPUT\_PAR

```
#define ERROR_INPUT_PAR -13
```

wrong input argument  
Definition at line 24 of file fasp\_const.h.

#### 9.43.2.24 ERROR\_LIC\_TYPE

```
#define ERROR_LIC_TYPE -80
```

wrong license type  
Definition at line 54 of file fasp\_const.h.

#### 9.43.2.25 ERROR\_MAT\_SIZE

```
#define ERROR_MAT_SIZE -15
```

wrong problem size  
Definition at line 26 of file fasp\_const.h.

#### 9.43.2.26 ERROR\_MISC

```
#define ERROR_MISC -19
```

other error  
Definition at line 28 of file fasp\_const.h.

#### 9.43.2.27 ERROR\_NUM\_BLOCKS

```
#define ERROR_NUM_BLOCKS -18
```

wrong number of blocks  
Definition at line 27 of file fasp\_const.h.

#### 9.43.2.28 ERROR\_OPEN\_FILE

```
#define ERROR_OPEN_FILE -10
```

fail to open a file  
Definition at line 22 of file fasp\_const.h.

#### 9.43.2.29 ERROR\_QUAD\_DIM

```
#define ERROR_QUAD_DIM -61
```

unsupported quadrature dim  
Definition at line 52 of file fasp\_const.h.

#### 9.43.2.30 ERROR\_QUAD\_TYPE

```
#define ERROR_QUAD_TYPE -60
```

unknown quadrature type  
Definition at line 51 of file fasp\_const.h.

#### 9.43.2.31 ERROR\_READ\_FILE

```
#define ERROR_READ_FILE -1
```

fail to read a file  
Definition at line 21 of file fasp\_const.h.

#### 9.43.2.32 ERROR\_REGRESS

```
#define ERROR_REGRESS -14
```

regression test fail  
Definition at line 25 of file fasp\_const.h.

#### 9.43.2.33 ERROR\_SOLVER\_EXIT

```
#define ERROR_SOLVER_EXIT -49
```

solver does not quit successfully  
Definition at line 49 of file fasp\_const.h.

#### 9.43.2.34 ERROR\_SOLVER\_ILUSETUP

```
#define ERROR_SOLVER_ILUSETUP -45
```

ILU setup error  
Definition at line 46 of file fasp\_const.h.

#### 9.43.2.35 ERROR\_SOLVER\_MAXIT

```
#define ERROR_SOLVER_MAXIT -48
```

maximal iteration number exceeded  
Definition at line 48 of file fasp\_const.h.

**9.43.2.36 ERROR\_SOLVER\_MISC**

#define ERROR\_SOLVER\_MISC -46  
misc solver error during run time  
Definition at line 47 of file fasp\_const.h.

**9.43.2.37 ERROR\_SOLVER\_PRECTYPE**

#define ERROR\_SOLVER\_PRECTYPE -41  
unknown precondition type  
Definition at line 42 of file fasp\_const.h.

**9.43.2.38 ERROR\_SOLVER\_SOLSTAG**

#define ERROR\_SOLVER\_SOLSTAG -43  
solver's solution is too small  
Definition at line 44 of file fasp\_const.h.

**9.43.2.39 ERROR\_SOLVER\_STAG**

#define ERROR\_SOLVER\_STAG -42  
solver stagnates  
Definition at line 43 of file fasp\_const.h.

**9.43.2.40 ERROR\_SOLVER\_TOLSMALL**

#define ERROR\_SOLVER\_TOLSMALL -44  
solver's tolerance is too small  
Definition at line 45 of file fasp\_const.h.

**9.43.2.41 ERROR\_SOLVER\_TYPE**

#define ERROR\_SOLVER\_TYPE -40  
unknown solver type  
Definition at line 41 of file fasp\_const.h.

**9.43.2.42 ERROR\_UNKNOWN**

#define ERROR\_UNKNOWN -99  
an unknown error type  
Definition at line 56 of file fasp\_const.h.

**9.43.2.43 ERROR\_WRONG\_FILE**

#define ERROR\_WRONG\_FILE -11  
input contains wrong format  
Definition at line 23 of file fasp\_const.h.

#### 9.43.2.44 FALSE

```
#define FALSE 0
```

logic FALSE  
Definition at line 62 of file fasp\_const.h.

#### 9.43.2.45 FASP\_SUCCESS

```
#define FASP_SUCCESS 0
```

Definition of return status and error messages.  
return from function successfully  
Definition at line 19 of file fasp\_const.h.

#### 9.43.2.46 FGPT

```
#define FGPT 0
```

Fine grid points  
  
Definition at line 226 of file fasp\_const.h.

#### 9.43.2.47 FPFIRST

```
#define FPFIRST -1
```

F-points first order  
Definition at line 241 of file fasp\_const.h.

#### 9.43.2.48 G0PT

```
#define G0PT -5
```

Type of vertices (DOFs) for coarsening.  
Cannot fit in aggregates  
Definition at line 224 of file fasp\_const.h.

#### 9.43.2.49 ILUk

```
#define ILUk 1
```

Type of ILU methods.  
ILUk  
Definition at line 148 of file fasp\_const.h.

#### 9.43.2.50 ILUt

```
#define ILUt 2
```

ILUt  
Definition at line 149 of file fasp\_const.h.

#### 9.43.2.51 ILUtp

```
#define ILUtp 3
```

ILUtp

Definition at line 150 of file fasp\_const.h.

#### 9.43.2.52 INTERP\_DIR

```
#define INTERP_DIR 1
```

Definition of interpolation types.

Direct interpolation

Definition at line 216 of file fasp\_const.h.

#### 9.43.2.53 INTERP\_ENG

```
#define INTERP_ENG 3
```

Energy minimization interpolation

Definition at line 218 of file fasp\_const.h.

#### 9.43.2.54 INTERP\_EXT

```
#define INTERP_EXT 6
```

Extended interpolation

Definition at line 219 of file fasp\_const.h.

#### 9.43.2.55 INTERP\_STD

```
#define INTERP_STD 2
```

Standard interpolation

Definition at line 217 of file fasp\_const.h.

#### 9.43.2.56 ISPT

```
#define ISPT 2
```

Isolated points

Definition at line 228 of file fasp\_const.h.

#### 9.43.2.57 MAT\_bBSR

```
#define MAT_bBSR 12
```

block BSR/CSR matrix

Definition at line 95 of file fasp\_const.h.

#### 9.43.2.58 MAT\_bCSR

```
#define MAT_bCSR 11
```

block CSR/CSR matrix == 2\*2 BLC matrix

Definition at line 94 of file fasp\_const.h.

#### 9.43.2.59 MAT\_BLC

```
#define MAT_BLC 8
```

block CSR matrix  
Definition at line 90 of file fasp\_const.h.

#### 9.43.2.60 MAT\_BSR

```
#define MAT_BSR 2
```

block-wise compressed sparse row  
Definition at line 86 of file fasp\_const.h.

#### 9.43.2.61 MAT\_bSTR

```
#define MAT_bSTR 13
```

block STR/CSR matrix  
Definition at line 96 of file fasp\_const.h.

#### 9.43.2.62 MAT\_CSR

```
#define MAT_CSR 1
```

compressed sparse row  
Definition at line 85 of file fasp\_const.h.

#### 9.43.2.63 MAT\_CSRL

```
#define MAT_CSRL 6
```

modified CSR to reduce cache missing  
Definition at line 88 of file fasp\_const.h.

#### 9.43.2.64 MAT\_FREE

```
#define MAT_FREE 0
```

Definition of matrix format.  
matrix-free format: only mxv action  
Definition at line 83 of file fasp\_const.h.

#### 9.43.2.65 MAT\_STR

```
#define MAT_STR 3
```

structured sparse matrix  
Definition at line 87 of file fasp\_const.h.

#### 9.43.2.66 MAT\_SymCSR

```
#define MAT_SymCSR 7
```

symmetric CSR format  
Definition at line 89 of file fasp\_const.h.



**9.43.2.67 MAX\_AMG\_LVL**

```
#define MAX_AMG_LVL 20
```

Maximal AMG coarsening level  
Definition at line 252 of file fasp\_const.h.

**9.43.2.68 MAX\_CRATE**

```
#define MAX_CRATE 20.0
```

Maximal coarsening ratio  
Definition at line 255 of file fasp\_const.h.

**9.43.2.69 MAX\_REFINE\_LVL**

```
#define MAX_REFINE_LVL 20
```

Maximal refinement level  
Definition at line 251 of file fasp\_const.h.

**9.43.2.70 MAX\_RESTART**

```
#define MAX_RESTART 20
```

Maximal restarting number  
Definition at line 256 of file fasp\_const.h.

**9.43.2.71 MAX\_STAG**

```
#define MAX_STAG 20
```

Maximal number of stagnation times  
Definition at line 257 of file fasp\_const.h.

**9.43.2.72 MIN\_CDOF**

```
#define MIN_CDOF 20
```

Minimal number of coarsest variables  
Definition at line 253 of file fasp\_const.h.

**9.43.2.73 MIN\_CRATE**

```
#define MIN_CRATE 0.9
```

Minimal coarsening ratio  
Definition at line 254 of file fasp\_const.h.

**9.43.2.74 NL\_AMLI\_CYCLE**

```
#define NL_AMLI_CYCLE 4
```

Nonlinear AMLI-cycle  
Definition at line 180 of file fasp\_const.h.

#### 9.43.2.75 NO\_ORDER

```
#define NO_ORDER 0
```

Definition of smoothing order.  
Natural order smoothing  
Definition at line 233 of file fasp\_const.h.

#### 9.43.2.76 NPAIR

```
#define NPAIR 3
```

non-symmetric pairwise aggregation  
Definition at line 171 of file fasp\_const.h.

#### 9.43.2.77 OFF

```
#define OFF 0
```

turn off certain parameter  
Definition at line 68 of file fasp\_const.h.

#### 9.43.2.78 ON

```
#define ON 1
```

Definition of switch.  
turn on certain parameter  
Definition at line 67 of file fasp\_const.h.

#### 9.43.2.79 OPENMP\_HOLDS

```
#define OPENMP_HOLDS 2000
```

Smallest size for OpenMP version  
Definition at line 259 of file fasp\_const.h.

#### 9.43.2.80 PAIRWISE

```
#define PAIRWISE 1
```

Definition of aggregation types.  
pairwise aggregation, default is SPAIR  
Definition at line 169 of file fasp\_const.h.

#### 9.43.2.81 PREC\_AMG

```
#define PREC_AMG 2
```

with AMG preconditioner  
Definition at line 140 of file fasp\_const.h.

#### 9.43.2.82 PREC\_DIAG

```
#define PREC_DIAG 1
```

with diagonal preconditioner

Definition at line 139 of file fasp\_const.h.

#### 9.43.2.83 PREC\_FMG

```
#define PREC_FMG 3
```

with full AMG precondition

Definition at line 141 of file fasp\_const.h.

#### 9.43.2.84 PREC\_ILU

```
#define PREC_ILU 4
```

with ILU precondition

Definition at line 142 of file fasp\_const.h.

#### 9.43.2.85 PREC\_NULL

```
#define PREC_NULL 0
```

Definition of preconditioner type for iterative methods.

with no precondition

Definition at line 138 of file fasp\_const.h.

#### 9.43.2.86 PREC\_SCHWARZ

```
#define PREC_SCHWARZ 5
```

with Schwarz preconditioner

Definition at line 143 of file fasp\_const.h.

#### 9.43.2.87 PRINT\_ALL

```
#define PRINT_ALL 10
```

all: all printouts, including files

Definition at line 78 of file fasp\_const.h.

#### 9.43.2.88 PRINT\_MIN

```
#define PRINT_MIN 1
```

quiet: print error, important warnings

Definition at line 74 of file fasp\_const.h.

#### 9.43.2.89 PRINT\_MORE

```
#define PRINT_MORE 4
```

more: print some useful debug info

Definition at line 76 of file fasp\_const.h.

#### 9.43.2.90 PRINT\_MOST

```
#define PRINT_MOST 8
```

most: maximal printouts, no files  
Definition at line 77 of file fasp\_const.h.

#### 9.43.2.91 PRINT\_NONE

```
#define PRINT_NONE 0
```

Print level for all subroutines – not including DEBUG output.  
silent: no printout at all  
Definition at line 73 of file fasp\_const.h.

#### 9.43.2.92 PRINT\_SOME

```
#define PRINT_SOME 2
```

some: print less important warnings  
Definition at line 75 of file fasp\_const.h.

#### 9.43.2.93 SA\_AMG

```
#define SA_AMG 2
```

smoothed aggregation AMG  
Definition at line 163 of file fasp\_const.h.

#### 9.43.2.94 SCHWARZ\_BACKWARD

```
#define SCHWARZ_BACKWARD 2
```

Backward ordering  
Definition at line 156 of file fasp\_const.h.

#### 9.43.2.95 SCHWARZ\_FORWARD

```
#define SCHWARZ_FORWARD 1
```

Type of Schwarz smoother.  
Forward ordering  
Definition at line 155 of file fasp\_const.h.

#### 9.43.2.96 SCHWARZ\_SYMMETRIC

```
#define SCHWARZ_SYMMETRIC 3
```

Symmetric smoother  
Definition at line 157 of file fasp\_const.h.

#### 9.43.2.97 SMALLREAL

```
#define SMALLREAL 1e-20
```

A small real number  
Definition at line 249 of file fasp\_const.h.

#### 9.43.2.98 SMALLREAL2

```
#define SMALLREAL2 1e-40
```

An extremely small real number

Definition at line 250 of file fasp\_const.h.

#### 9.43.2.99 SMOOTHER\_BLKOil

```
#define SMOOTHER_BLKOil 11
```

Definition of specialized smoother types.

Used in monolithic AMG for black-oil

Definition at line 201 of file fasp\_const.h.

#### 9.43.2.100 SMOOTHER\_CG

```
#define SMOOTHER_CG 4
```

CG as a smoother

Definition at line 190 of file fasp\_const.h.

#### 9.43.2.101 SMOOTHER\_GS

```
#define SMOOTHER_GS 2
```

Gauss-Seidel smoother

Definition at line 188 of file fasp\_const.h.

#### 9.43.2.102 SMOOTHER\_GSOR

```
#define SMOOTHER_GSOR 7
```

GS + SOR smoother

Definition at line 193 of file fasp\_const.h.

#### 9.43.2.103 SMOOTHER\_JACOBI

```
#define SMOOTHER_JACOBI 1
```

Definition of standard smoother types.

Jacobi smoother

Definition at line 187 of file fasp\_const.h.

#### 9.43.2.104 SMOOTHER\_L1DIAG

```
#define SMOOTHER_L1DIAG 10
```

L1 norm diagonal scaling smoother

Definition at line 196 of file fasp\_const.h.

**9.43.2.105 SMOOTHER\_POLY**

```
#define SMOOTHER_POLY 9
```

Polynomial smoother  
Definition at line 195 of file fasp\_const.h.

**9.43.2.106 SMOOTHER\_SGS**

```
#define SMOOTHER_SGS 3
```

Symmetric Gauss-Seidel smoother  
Definition at line 189 of file fasp\_const.h.

**9.43.2.107 SMOOTHER\_SGSOR**

```
#define SMOOTHER_SGSOR 8
```

SGS + SSOR smoother  
Definition at line 194 of file fasp\_const.h.

**9.43.2.108 SMOOTHER\_SOR**

```
#define SMOOTHER_SOR 5
```

SOR smoother  
Definition at line 191 of file fasp\_const.h.

**9.43.2.109 SMOOTHER\_SPETEN**

```
#define SMOOTHER_SPETEN 19
```

Used in monolithic AMG for black-oil  
Definition at line 202 of file fasp\_const.h.

**9.43.2.110 SMOOTHER\_SSOR**

```
#define SMOOTHER_SSOR 6
```

SSOR smoother  
Definition at line 192 of file fasp\_const.h.

**9.43.2.111 SOLVER\_AMG**

```
#define SOLVER_AMG 21
```

AMG as an iterative solver  
Definition at line 120 of file fasp\_const.h.

**9.43.2.112 SOLVER\_BiCGstab**

```
#define SOLVER_BiCGstab 2
```

Bi-Conjugate Gradient Stabilized  
Definition at line 104 of file fasp\_const.h.

**9.43.2.113 SOLVER\_CG**

```
#define SOLVER_CG 1
```

Conjugate Gradient  
Definition at line 103 of file fasp\_const.h.

**9.43.2.114 SOLVER\_DEFAULT**

```
#define SOLVER_DEFAULT 0
```

Definition of solver types for iterative methods.  
Use default solver in FASP  
Definition at line 101 of file fasp\_const.h.

**9.43.2.115 SOLVER\_FMG**

```
#define SOLVER_FMG 22
```

Full AMG as an solver  
Definition at line 121 of file fasp\_const.h.

**9.43.2.116 SOLVER\_GCG**

```
#define SOLVER_GCG 7
```

Generalized Conjugate Gradient  
Definition at line 109 of file fasp\_const.h.

**9.43.2.117 SOLVER\_GCR**

```
#define SOLVER_GCR 8
```

Generalized Conjugate Residual  
Definition at line 110 of file fasp\_const.h.

**9.43.2.118 SOLVER\_GMRES**

```
#define SOLVER_GMRES 4
```

Generalized Minimal Residual  
Definition at line 106 of file fasp\_const.h.

**9.43.2.119 SOLVER\_MinRes**

```
#define SOLVER_MinRes 3
```

Minimal Residual  
Definition at line 105 of file fasp\_const.h.

**9.43.2.120 SOLVER\_MUMPS**

```
#define SOLVER_MUMPS 33
```

Direct Solver: MUMPS  
Definition at line 125 of file fasp\_const.h.

**9.43.2.121 SOLVER\_PARDISO**

```
#define SOLVER_PARDISO 34
```

Direct Solver: PARDISO  
Definition at line 126 of file fasp\_const.h.

**9.43.2.122 SOLVER\_SBiCGstab**

```
#define SOLVER_SBiCGstab 12
```

BiCGstab with safety net  
Definition at line 113 of file fasp\_const.h.

**9.43.2.123 SOLVER\_SCG**

```
#define SOLVER_SCG 11
```

Conjugate Gradient with safety net  
Definition at line 112 of file fasp\_const.h.

**9.43.2.124 SOLVER\_SGCG**

```
#define SOLVER_SGCG 17
```

GCG with safety net  
Definition at line 118 of file fasp\_const.h.

**9.43.2.125 SOLVER\_SGMRES**

```
#define SOLVER_SGMRES 14
```

GMRes with safety net  
Definition at line 115 of file fasp\_const.h.

**9.43.2.126 SOLVER\_SMinRes**

```
#define SOLVER_SMinRes 13
```

MinRes with safety net  
Definition at line 114 of file fasp\_const.h.

**9.43.2.127 SOLVER\_SUPERLU**

```
#define SOLVER_SUPERLU 31
```

Direct Solver: SuperLU  
Definition at line 123 of file fasp\_const.h.

**9.43.2.128 SOLVER\_SVFGMRES**

```
#define SOLVER_SVFGMRES 16
```

Variable-restart FGMRES with safety net  
Definition at line 117 of file fasp\_const.h.



**9.43.2.129 SOLVER\_SVGMRES**

#define SOLVER\_SVGMRES 15  
Variable-restart GMRES with safety net  
Definition at line 116 of file fasp\_const.h.

**9.43.2.130 SOLVER\_UMFPACK**

#define SOLVER\_UMFPACK 32  
Direct Solver: UMFPack  
Definition at line 124 of file fasp\_const.h.

**9.43.2.131 SOLVER\_VFGMRES**

#define SOLVER\_VFGMRES 6  
Variable Restarting Flexible GMRES  
Definition at line 108 of file fasp\_const.h.

**9.43.2.132 SOLVER\_VGMRES**

#define SOLVER\_VGMRES 5  
Variable Restarting GMRES  
Definition at line 107 of file fasp\_const.h.

**9.43.2.133 SPAIR**

#define SPAIR 4  
symmetric pairwise aggregation  
Definition at line 172 of file fasp\_const.h.

**9.43.2.134 STAG\_RATIO**

#define STAG\_RATIO 1e-4  
Stagnation tolerance = tol\*STAGRATIO  
Definition at line 258 of file fasp\_const.h.

**9.43.2.135 STOP\_MOD\_REL\_RES**

#define STOP\_MOD\_REL\_RES 3  
modified relative residual  $\|r\|/\|x\|$   
Definition at line 133 of file fasp\_const.h.

**9.43.2.136 STOP\_REL\_PRECRES**

#define STOP\_REL\_PRECRES 2  
relative B-residual  $\|r\|_B/\|b\|_B$   
Definition at line 132 of file fasp\_const.h.

**9.43.2.137 STOP\_REL\_RES**

```
#define STOP_REL_RES 1
```

Definition of iterative solver stopping criteria types.

relative residual  $\|r\|/\|b\|$

Definition at line 131 of file fasp\_const.h.

**9.43.2.138 TRUE**

```
#define TRUE 1
```

Definition of logic type.

logic TRUE

Definition at line 61 of file fasp\_const.h.

**9.43.2.139 UA\_AMG**

```
#define UA_AMG 3
```

unsmoothed aggregation AMG

Definition at line 164 of file fasp\_const.h.

**9.43.2.140 UNPT**

```
#define UNPT -1
```

Undetermined points

Definition at line 225 of file fasp\_const.h.

**9.43.2.141 USERDEFINED**

```
#define USERDEFINED 0
```

Type of ordering for smoothers.

User defined order

Definition at line 239 of file fasp\_const.h.

**9.43.2.142 V\_CYCLE**

```
#define V_CYCLE 1
```

Definition of cycle types.

V-cycle

Definition at line 177 of file fasp\_const.h.

**9.43.2.143 VMB**

```
#define VMB 2
```

VMB aggregation

Definition at line 170 of file fasp\_const.h.

#### 9.43.2.144 VW\_CYCLE

```
#define VW_CYCLE 12
```

VW-cycle

Definition at line 181 of file fasp\_const.h.

#### 9.43.2.145 W\_CYCLE

```
#define W_CYCLE 2
```

W-cycle

Definition at line 178 of file fasp\_const.h.

#### 9.43.2.146 WV\_CYCLE

```
#define WV_CYCLE 21
```

WV-cycle

Definition at line 182 of file fasp\_const.h.

## 9.44 fasp\_grid.h File Reference

Header file for FASP grid.

### Data Structures

- struct [grid2d](#)  
*Two dimensional grid data structure.*

### Macros

- #define [\\_\\_FASPGRID\\_HEADER\\_\\_](#)

### Typedefs

- typedef struct [grid2d](#) [grid2d](#)
- typedef [grid2d](#) \* [pgrid2d](#)
- typedef const [grid2d](#) \* [pcgrid2d](#)

#### 9.44.1 Detailed Description

Header file for FASP grid.

Copyright (C) 2015–2020 by the FASP team. All rights reserved.

---

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.44.2 Macro Definition Documentation

##### 9.44.2.1 [\\_\\_FASPGRID\\_HEADER\\_\\_](#)

```
#define \_\_FASPGRID\_HEADER\_\_
```

indicate [fasp\\_grid.h](#) has been included before

Definition at line 12 of file fasp\_grid.h.

### 9.44.3 Typedef Documentation

#### 9.44.3.1 grid2d

typedef struct [grid2d](#) [grid2d](#)  
2D grid type for plotting

#### 9.44.3.2 pcgrid2d

typedef const [grid2d](#)\* [pcgrid2d](#)  
Grid in 2d  
Definition at line 45 of file [fasp\\_grid.h](#).

#### 9.44.3.3 pgrid2d

typedef [grid2d](#)\* [pgrid2d](#)  
Grid in 2d  
Definition at line 43 of file [fasp\\_grid.h](#).

## 9.45 ItrSmootherBSR.c File Reference

Smoothers for [dBSRmat](#) matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_smoother\\_dbsr\\_jacobi](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u)  
*Jacobi relaxation.*
- void [fasp\\_smoother\\_dbsr\\_jacobi\\_setup](#) ([dBSRmat](#) \*A, [REAL](#) \*diaginv)  
*Setup for jacobi relaxation, fetch the diagonal sub-block matrixes and make them inverse first.*
- void [fasp\\_smoother\\_dbsr\\_jacobi1](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv)  
*Jacobi relaxation.*
- void [fasp\\_smoother\\_dbsr\\_gs](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [INT](#) order, [INT](#) \*mark)  
*Gauss-Seidel relaxation.*
- void [fasp\\_smoother\\_dbsr\\_gs1](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [INT](#) order, [INT](#) \*mark, [REAL](#) \*diaginv)  
*Gauss-Seidel relaxation.*
- void [fasp\\_smoother\\_dbsr\\_gs\\_ascend](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv)  
*Gauss-Seidel relaxation in the ascending order.*
- void [fasp\\_smoother\\_dbsr\\_gs\\_ascend1](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u)  
*Gauss-Seidel relaxation in the ascending order.*
- void [fasp\\_smoother\\_dbsr\\_gs\\_descend](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv)  
*Gauss-Seidel relaxation in the descending order.*
- void [fasp\\_smoother\\_dbsr\\_gs\\_descend1](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u)  
*Gauss-Seidel relaxation in the descending order.*
- void [fasp\\_smoother\\_dbsr\\_gs\\_order1](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [INT](#) \*mark)

*Gauss-Seidel relaxation in the user-defined order.*

- void `fasp_smoother_dbsr_gs_order2` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `INT` \*mark, `REAL` \*work)

*Gauss-Seidel relaxation in the user-defined order.*

- void `fasp_smoother_dbsr_sor` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `INT` order, `INT` \*mark, `REAL` weight)

*SOR relaxation.*

- void `fasp_smoother_dbsr_sor1` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `INT` order, `INT` \*mark, `REAL` \*diaginv, `REAL` weight)

*SOR relaxation.*

- void `fasp_smoother_dbsr_sor_ascend` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `REAL` \*diaginv, `REAL` weight)

*SOR relaxation in the ascending order.*

- void `fasp_smoother_dbsr_sor_descend` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `REAL` \*diaginv, `REAL` weight)

*SOR relaxation in the descending order.*

- void `fasp_smoother_dbsr_sor_order` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*u, `REAL` \*diaginv, `INT` \*mark, `REAL` weight)

*SOR relaxation in the user-defined order.*

- void `fasp_smoother_dbsr_ilu` (`dBSRmat` \*A, `dvector` \*b, `dvector` \*x, void \*data)

*ILU method as the smoother in solving  $Au=b$  with multigrid method.*

## Variables

- `REAL` `ilu_solve_time` = 0.0

### 9.45.1 Detailed Description

Smoothers for `dBSRmat` matrices.

Note

This file contains Level-2 (ltr) functions. It requires: `AuxArray.c`, `AuxMemory.c`, `AuxMessage.c`, `AuxThreads.c`, `AuxTiming.c`, `BlaSmallMatInv.c`, `BlaSmallMat.c`, `BlaArray.c`, `BlaSpmvBSR.c`, and `PreBSR.c`

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

// TODO: Need to optimize routines here! –Chensong

### 9.45.2 Function Documentation

#### 9.45.2.1 `fasp_smoother_dbsr_gs()`

```
void fasp_smoother_dbsr_gs (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    INT order,
    INT * mark )
```

Gauss-Seidel relaxation.

Parameters

A	Pointer to <code>dBSRmat</code> : the coefficient matrix
---	--

## Parameters

<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending order DESCEND 21: in descending order If mark != NULL: in the user-defined order
<i>mark</i>	Pointer to NULL or to the user-defined ordering

## Author

Zhiyang Zhou

## Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/03/2012  
Definition at line 428 of file ltrSmootherBSR.c.

**9.45.2.2 fasp\_smoother\_dbsr\_gs1()**

```
void fasp_smoother_dbsr_gs1 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    INT order,
    INT * mark,
    REAL * diaginv )
```

Gauss-Seidel relaxation.

## Parameters

<i>A</i>	Pointer to <b>dBSRmat</b> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending order DESCEND 21: in descending order If mark != NULL: in the user-defined order
<i>mark</i>	Pointer to NULL or to the user-defined ordering
<i>diaginv</i>	Inverses for all the diagonal blocks of A

## Author

Zhiyang Zhou

## Date

2010/10/25

Definition at line 545 of file ltrSmootherBSR.c.

**9.45.2.3 fasp\_smoother\_dbsr\_gs\_ascend()**

```
void fasp_smoother_dbsr_gs_ascend (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv )
```

Gauss-Seidel relaxation in the ascending order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of <i>A</i>

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Definition at line 582 of file ltrSmootherBSR.c.

**9.45.2.4 fasp\_smoother\_dbsr\_gs\_ascend1()**

```
void fasp_smoother_dbsr_gs_ascend1 (
    dBSRmat * A,
    dvector * b,
    dvector * u )
```

Gauss-Seidel relaxation in the ascending order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)

**Author**

Xiaozhe Hu

**Date**

01/01/2014

**Note**

The only difference between the functions 'fasp\_smoother\_dbsr\_gs\_ascend1' and 'fasp\_smoother\_dbsr\_gs\_↔ascend' is that we don't have to multiply by the inverses of the diagonal blocks in each ROW since matrix *A* has been such scaled that all the diagonal blocks become identity matrices.

Definition at line 655 of file ltrSmootherBSR.c.

### 9.45.2.5 fasp\_smoother\_dbsr\_gs\_descend()

```
void fasp_smoother_dbsr_gs_descend (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginvs )
```

Gauss-Seidel relaxation in the descending order.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A

#### Author

Zhiyang Zhou

#### Date

2010/10/25

Definition at line 724 of file ltrSmootherBSR.c.

### 9.45.2.6 fasp\_smoother\_dbsr\_gs\_descend1()

```
void fasp_smoother_dbsr_gs_descend1 (
    dBSRmat * A,
    dvector * b,
    dvector * u )
```

Gauss-Seidel relaxation in the descending order.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)

#### Author

Xiaozhe Hu

#### Date

01/01/2014

#### Note

The only difference between the functions 'fasp\_smoother\_dbsr\_gs\_ascend1' and 'fasp\_smoother\_dbsr\_gs\_↔ascend' is that we don't have to multiply by the inverses of the diagonal blocks in each ROW since matrix A has been such scaled that all the diagonal blocks become identity matrices.

Definition at line 798 of file ltrSmootherBSR.c.



**9.45.2.7 fasp\_smoother\_dbsr\_gs\_order1()**

```
void fasp_smoother_dbsr_gs_order1 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    INT * mark )
```

Gauss-Seidel relaxation in the user-defined order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>mark</i>	Pointer to the user-defined ordering

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Definition at line 868 of file ltrSmootherBSR.c.

**9.45.2.8 fasp\_smoother\_dbsr\_gs\_order2()**

```
void fasp_smoother_dbsr_gs_order2 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    INT * mark,
    REAL * work )
```

Gauss-Seidel relaxation in the user-defined order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>mark</i>	Pointer to the user-defined ordering
<i>work</i>	Work temp array

**Author**

Zhiyang Zhou

**Date**

2010/11/08

**Note**

The only difference between the functions 'fasp\_smoother\_dbsr\_gs\_order2' and 'fasp\_smoother\_dbsr\_gs\_order1' lies in that we don't have to multiply by the inverses of the diagonal blocks in each ROW since matrix A has been such scaled that all the diagonal blocks become identity matrices.

Definition at line 946 of file ltrSmootherBSR.c.

**9.45.2.9 fasp\_smoother\_dbsr\_ilu()**

```
void fasp_smoother_dbsr_ilu (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    void * data )
```

ILU method as the smoother in solving  $Au=b$  with multigrid method.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>data</i>	Pointer to user defined data

**Author**

Zhiyang Zhou, Zheng Li

**Date**

2010/10/25

NOTE: Add multi-threads parallel ILU block by Zheng Li 12/04/2016. form residual  $zr = b - A x$  solve LU  $z=zr$

$x=x+z$

Definition at line 1566 of file ltrSmootherBSR.c.

**9.45.2.10 fasp\_smoother\_dbsr\_jacobi()**

```
void fasp_smoother_dbsr_jacobi (
    dBSRmat * A,
    dvector * b,
    dvector * u )
```

Jacobi relaxation.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/02/2012  
 Definition at line 59 of file ltrSmootherBSR.c.

**9.45.2.11 fasp\_smoother\_dbsr\_jacobi1()**

```
void fasp_smoother_dbsr_jacobi1 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv )
```

Jacobi relaxation.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/03/2012  
 Definition at line 274 of file ltrSmootherBSR.c.

**9.45.2.12 fasp\_smoother\_dbsr\_jacobi\_setup()**

```
void fasp_smoother_dbsr_jacobi_setup (
    dBSRmat * A,
    REAL * diaginv )
```

Setup for jacobi relaxation, fetch the diagonal sub-block matrixes and make them inverse first.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>diaginv</i>	Inverse of the diagonal entries

**Author**

Zhiyang Zhou

**Date**

10/25/2010

Modified by Chunsheng Feng, Zheng Li on 08/02/2012  
 Definition at line 168 of file ltrSmootherBSR.c.

**9.45.2.13 fasp\_smoother\_dbsr\_sor()**

```
void fasp_smoother_dbsr_sor (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    INT order,
    INT * mark,
    REAL weight )
```

SOR relaxation.

**Parameters**

<i>A</i>	Pointer to <b>dBSRmat</b> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending order DESCEND 21: in descending order If mark != NULL: in the user-defined order
<i>mark</i>	Pointer to NULL or to the user-defined ordering
<i>weight</i>	Over-relaxation weight

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/03/2012  
 Definition at line 1023 of file ltrSmootherBSR.c.

**9.45.2.14 fasp\_smoother\_dbsr\_sor1()**

```
void fasp_smoother_dbsr_sor1 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    INT order,
    INT * mark,
    REAL * diaginv,
    REAL weight )
```

SOR relaxation.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending order DESCEND 21: in descending order If mark != NULL: in the user-defined order
<i>mark</i>	Pointer to NULL or to the user-defined ordering
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>weight</i>	Over-relaxation weight

## Author

Zhiyang Zhou

## Date

2010/10/25

Definition at line 1146 of file ltrSmootherBSR.c.

**9.45.2.15 fasp\_smoother\_dbsr\_sor\_ascend()**

```
void fasp_smoother_dbsr_sor_ascend (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    REAL weight )
```

SOR relaxation in the ascending order.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>weight</i>	Over-relaxation weight

## Author

Zhiyang Zhou

## Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 2012/09/04

Definition at line 1187 of file ltrSmootherBSR.c.

**9.45.2.16 fasp\_smoother\_dbsr\_sor\_descend()**

```
void fasp_smoother_dbsr_sor_descend (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginvs,
    REAL weight )
```

SOR relaxation in the descending order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>weight</i>	Over-relaxation weight

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 2012/09/04  
Definition at line 1310 of file ltrSmootherBSR.c.

**9.45.2.17 fasp\_smoother\_dbsr\_sor\_order()**

```
void fasp_smoother_dbsr_sor_order (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginvs,
    INT * mark,
    REAL weight )
```

SOR relaxation in the user-defined order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>mark</i>	Pointer to the user-defined ordering
<i>weight</i>	Over-relaxation weight

## Author

Zhiyang Zhou

## Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 2012/09/04  
 Definition at line 1438 of file ltrSmootherBSR.c.

### 9.45.3 Variable Documentation

#### 9.45.3.1 ilu\_solve\_time

```
REAL ilu_solve_time = 0.0
```

ILU time for the SOLVE phase

Definition at line 39 of file ltrSmootherBSR.c.

## 9.46 ltrSmootherCSR.c File Reference

Smoothers for dCSRmat matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void `fasp_smoother_dcsr_jacobi` (dvector \*u, const INT i\_1, const INT i\_n, const INT s, dCSRmat \*A, dvector \*b, INT L, const REAL w)  
*Weighted Jacobi method as a smoother.*
- void `fasp_smoother_dcsr_gs` (dvector \*u, const INT i\_1, const INT i\_n, const INT s, dCSRmat \*A, dvector \*b, INT L)  
*Gauss-Seidel method as a smoother.*
- void `fasp_smoother_dcsr_gs_cf` (dvector \*u, dCSRmat \*A, dvector \*b, INT L, INT \*mark, const INT order)  
*Gauss-Seidel smoother with C/F ordering for Au=b.*
- void `fasp_smoother_dcsr_sgs` (dvector \*u, dCSRmat \*A, dvector \*b, INT L)  
*Symmetric Gauss-Seidel method as a smoother.*
- void `fasp_smoother_dcsr_sor` (dvector \*u, const INT i\_1, const INT i\_n, const INT s, dCSRmat \*A, dvector \*b, INT L, const REAL w)  
*SOR method as a smoother.*
- void `fasp_smoother_dcsr_sor_cf` (dvector \*u, dCSRmat \*A, dvector \*b, INT L, const REAL w, INT \*mark, const INT order)  
*SOR smoother with C/F ordering for Au=b.*
- void `fasp_smoother_dcsr_ilu` (dCSRmat \*A, dvector \*b, dvector \*x, void \*data)  
*ILU method as a smoother.*
- void `fasp_smoother_dcsr_kaczmarz` (dvector \*u, const INT i\_1, const INT i\_n, const INT s, dCSRmat \*A, dvector \*b, INT L, const REAL w)  
*Kaczmarz method as a smoother.*

- void `fasp_smoother_dcsr_L1diag` (`dvector` \*u, const `INT` i\_1, const `INT` i\_n, const `INT` s, `dCSRmat` \*A, `dvector` \*b, `INT` L)

*Diagonal scaling (using L1 norm) as a smoother.*

### 9.46.1 Detailed Description

Smoothers for `dCSRmat` matrices.

Note

This file contains Level-2 (ltr) functions. It requires: `AuxArray.c`, `AuxMemory.c`, `AuxMessage.c`, `AuxThreads.c`, `BlaArray.c`, and `BlaSpmvCSR.c`

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.46.2 Function Documentation

#### 9.46.2.1 `fasp_smoother_dcsr_gs()`

```
void fasp_smoother_dcsr_gs (
    dvector * u,
    const INT i_1,
    const INT i_n,
    const INT s,
    dCSRmat * A,
    dvector * b,
    INT L )
```

Gauss-Seidel method as a smoother.

Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i</i> <sub>←</sub> <i>←</i> <i>1</i>	Starting index
<i>i</i> <sub>←</sub> <i>←</i> <i>n</i>	Ending index
<i>s</i>	Increasing step
<i>A</i>	Pointer to <code>dBSRmat</code> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations

Author

Xuehai Huang, Chensong Zhang



## Date

09/26/2009

Modified by Chunsheng Feng, Zheng Li on 09/01/2012  
 Definition at line 190 of file ltrSmootherCSR.c.

**9.46.2.2 fasp\_smoother\_dcsr\_gs\_cf()**

```
void fasp_smoother_dcsr_gs_cf (
    dvector * u,
    dCSRmat * A,
    dvector * b,
    INT L,
    INT * mark,
    const INT order )
```

Gauss-Seidel smoother with C/F ordering for  $Au=b$ .

## Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations
<i>mark</i>	C/F marker array
<i>order</i>	C/F ordering: -1: F-first; 1: C-first

## Author

Zhiyang Zhou

## Date

11/12/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/24/2012  
 Definition at line 363 of file ltrSmootherCSR.c.

**9.46.2.3 fasp\_smoother\_dcsr\_ilu()**

```
void fasp_smoother_dcsr_ilu (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    void * data )
```

ILU method as a smoother.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>data</i>	Pointer to user defined data

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

2010/11/12

form residual  $zr = b - A x$ 

Definition at line 1065 of file ltrSmootherCSR.c.

**9.46.2.4 fasp\_smoother\_dcsr\_jacobi()**

```
void fasp_smoother_dcsr_jacobi (
    dvector * u,
    const INT i_1,
    const INT i_n,
    const INT s,
    dCSRmat * A,
    dvector * b,
    INT L,
    const REAL w )
```

Weighted Jacobi method as a smoother.

**Parameters**

$u$	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$i_{\leftarrow}$ $\_ \leftarrow$ $1$	Starting index
$i_{\leftarrow}$ $\_ \leftarrow$ $n$	Ending index
$s$	Increasing step
$A$	Pointer to dBSRmat: the coefficient matrix
$b$	Pointer to dvector: the right hand side
$L$	Number of iterations
$w$	Over-relaxation weight

**Author**

Xuehai Huang, Chensong Zhang

**Date**

09/26/2009

Modified by Chunsheng Feng, Zheng Li on 08/29/2012 Modified by Chensong Zhang on 08/24/2017: Pass weight  $w$  as a parameter

Definition at line 50 of file ltrSmootherCSR.c.

**9.46.2.5 fasp\_smoother\_dcsr\_kaczmarz()**

```
void fasp_smoother_dcsr_kaczmarz (
```

```

dvector * u,
const INT i_l,
const INT i_n,
const INT s,
dCSRmat * A,
dvector * b,
INT L,
const REAL w )

```

Kaczmarz method as a smoother.

#### Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i</i> <sub>↔</sub> <i>←</i> 1	Starting index
<i>i</i> <sub>↔</sub> <i>←</i> <i>n</i>	Ending index
<i>s</i>	Increasing step
<i>A</i>	Pointer to dCSRmat: the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations
<i>w</i>	Over-relaxation weight

#### Author

Xiaozhe Hu

#### Date

2010/11/12

Modified by Chunsheng Feng, Zheng Li on 2012/09/01

Definition at line 1144 of file ltrSmootherCSR.c.

#### 9.46.2.6 fasp\_smoother\_dcsr\_L1diag()

```

void fasp_smoother_dcsr_L1diag (
    dvector * u,
    const INT i_l,
    const INT i_n,
    const INT s,
    dCSRmat * A,
    dvector * b,
    INT L )

```

Diagonal scaling (using L1 norm) as a smoother.

#### Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i</i> <sub>↔</sub> <i>←</i> 1	Starting index

## Parameters

$i_{\leftarrow}$ $_{\leftarrow}$ $n$	Ending index
$s$	Increasing step
$A$	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
$b$	Pointer to dvector: the right hand side
$L$	Number of iterations

## Author

Xiaozhe Hu, James Brannick

## Date

01/26/2011

Modified by Chunsheng Feng, Zheng Li on 09/01/2012

Definition at line 1284 of file ltrSmootherCSR.c.

**9.46.2.7 fasp\_smoother\_dcsr\_sgs()**

```
void fasp_smoother_dcsr_sgs (
    dvector * u,
    dCSRmat * A,
    dvector * b,
    INT L )
```

Symmetric Gauss-Seidel method as a smoother.

## Parameters

$u$	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$A$	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
$b$	Pointer to dvector: the right hand side
$L$	Number of iterations

## Author

Xiaozhe Hu

## Date

10/26/2010

Modified by Chunsheng Feng, Zheng Li on 09/01/2012

Definition at line 628 of file ltrSmootherCSR.c.

**9.46.2.8 fasp\_smoother\_dcsr\_sor()**

```
void fasp_smoother_dcsr_sor (
    dvector * u,
```

```

const INT i_l,
const INT i_n,
const INT s,
dCSRmat * A,
dvector * b,
INT L,
const REAL w )

```

SOR method as a smoother.

#### Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i</i> ↔ ↔ 1	Starting index
<i>i</i> ↔ ↔ <i>n</i>	Ending index
<i>s</i>	Increasing step
<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations
<i>w</i>	Over-relaxation weight

#### Author

Xiaozhe Hu

#### Date

10/26/2010

Modified by Chunsheng Feng, Zheng Li on 09/01/2012  
Definition at line 744 of file ltrSmootherCSR.c.

#### 9.46.2.9 fasp\_smoother\_dcsr\_sor\_cf()

```

void fasp_smoother_dcsr_sor_cf (
    dvector * u,
    dCSRmat * A,
    dvector * b,
    INT L,
    const REAL w,
    INT * mark,
    const INT order )

```

SOR smoother with C/F ordering for Au=b.

#### Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations

**Parameters**

<i>w</i>	Over-relaxation weight
<i>mark</i>	C/F marker array
<i>order</i>	C/F ordering: -1: F-first; 1: C-first

**Author**

Zhiyang Zhou

**Date**

2010/11/12

Modified by Chunsheng Feng, Zheng Li on 08/29/2012  
 Definition at line 871 of file ltrSmootherCSR.c.

**9.47 ltrSmootherCSR.c File Reference**

Smoothers for [dCSRmat](#) matrices using compatible relaxation.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void [fasp\\_smoother\\_dcsr\\_gscr](#) (INT pt, INT n, REAL \*u, INT \*ia, INT \*ja, REAL \*a, REAL \*b, INT L, INT \*CF)  
*Gauss Seidel method restricted to a block.*

**9.47.1 Detailed Description**

Smoothers for [dCSRmat](#) matrices using compatible relaxation.

**Note**

Restricted smoothers for compatible relaxation, C/F smoothing, etc.

This file contains Level-2 (ltr) functions. It requires: [AuxMessage.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

// TODO: Need to optimize routines here! –Chensong

**9.47.2 Function Documentation**

**9.47.2.1 fasp\_smoother\_dcsr\_gscr()**

```
void fasp_smoother_dcsr_gscr (
    INT pt,
    INT n,
    REAL * u,
    INT * ia,
    INT * ja,
    REAL * a,
    REAL * b,
    INT L,
    INT * CF )
```

Gauss Seidel method restricted to a block.

**Parameters**

<i>pt</i>	Relax type, e.g., cpt, fpt, etc..
<i>n</i>	Number of variables
<i>u</i>	Iterated solution
<i>ia</i>	Row pointer
<i>ja</i>	Column index
<i>a</i>	Pointers to sparse matrix values in CSR format
<i>b</i>	Pointer to right hand side
<i>L</i>	Number of iterations
<i>CF</i>	Marker for C, F points

**Author**

James Brannick

**Date**

09/07/2010

**Note**

Gauss Seidel CR smoother (Smoother\_Type = 99)

Definition at line 48 of file ItrSmootherCSRcr.c.

**9.48 ItrSmootherCSRpoly.c File Reference**

Smoothers for [dCSRmat](#) matrices using poly. approx. to  $A^{-1}$ .

```
#include <math.h>
#include <time.h>
#include <float.h>
#include <limits.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void [fasp\\_smoother\\_dcsr\\_poly](#) ([dCSRmat](#) \*Amat, [dvector](#) \*brhs, [dvector](#) \*usol, INT n, INT ndeg, INT L)

*poly approx to  $A^{-1}$  as MG smoother*

- void `fasp_smoother_dcsr_poly_old` (`dCSRmat` \*Amat, `dvector` \*brhs, `dvector` \*usol, `INT` n, `INT` ndeg, `INT` L)

*poly approx to  $A^{-1}$  as MG smoother: JK&LTZ2010*

### 9.48.1 Detailed Description

Smoothers for `dCSRmat` matrices using poly. approx. to  $A^{-1}$ .

#### Note

This file contains Level-2 (ltr) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [BlaArray.c](#), and [BlaSpmvCSR.c](#)

Reference: Johannes K. Kraus, Panayot S. Vassilevski, Ludmil T. Zikatanov Polynomial of best uniform approximation to  $x^{-1}$  and smoothing in two-level methods, 2013.

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

#### Warning

Do NOT use auto-indentation in this file!

// TODO: Need to optimize routines here! –Chensong

### 9.48.2 Function Documentation

#### 9.48.2.1 `fasp_smoother_dcsr_poly()`

```
void fasp_smoother_dcsr_poly (
    dCSRmat * Amat,
    dvector * brhs,
    dvector * usol,
    INT n,
    INT ndeg,
    INT L )
```

poly approx to  $A^{-1}$  as MG smoother

#### Parameters

<i>Amat</i>	Pointer to stiffness matrix, consider square matrix.
<i>brhs</i>	Pointer to right hand side
<i>usol</i>	Pointer to solution
<i>n</i>	Problem size
<i>ndeg</i>	Degree of poly
<i>L</i>	Number of iterations

#### Author

Fei Cao, Xiaozhe Hu



## Date

05/24/2012

Definition at line 67 of file ItrSmootherCSRpoly.c.

**9.48.2.2 fasp\_smoother\_dcsr\_poly\_old()**

```
void fasp_smoother_dcsr_poly_old (
    dCSRmat * Amat,
    dvector * brhs,
    dvector * usol,
    INT n,
    INT ndeg,
    INT L )
```

poly approx to  $A^{-1}$  as MG smoother: JK&LTZ2010

## Parameters

<i>Amat</i>	Pointer to stiffness matrix
<i>brhs</i>	Pointer to right hand side
<i>usol</i>	Pointer to solution
<i>n</i>	Problem size
<i>ndeg</i>	Degree of poly
<i>L</i>	Number of iterations

## Author

James Brannick and Ludmil T Zikatanov

## Date

06/28/2010

Modified by Chunsheng Feng, Zheng Li on 10/18/2012

Definition at line 165 of file ItrSmootherCSRpoly.c.

**9.49 ItrSmootherSTR.c File Reference**Smoothers for **dSTRmat** matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void **fasp\_smoother\_dstr\_jacobi** (**dSTRmat** \*A, **dvector** \*b, **dvector** \*u)  
*Jacobi method as the smoother.*
- void **fasp\_smoother\_dstr\_jacobi1** (**dSTRmat** \*A, **dvector** \*b, **dvector** \*u, **REAL** \*diaginv)  
*Jacobi method as the smoother with diag\_inv given.*
- void **fasp\_smoother\_dstr\_gs** (**dSTRmat** \*A, **dvector** \*b, **dvector** \*u, const **INT** order, **INT** \*mark)

*Gauss-Seidel method as the smoother.*

- void [fasp\\_smoother\\_dstr\\_gs1](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, const [INT](#) order, [INT](#) \*mark, [REAL](#) \*diaginv)

*Gauss-Seidel method as the smoother with diag\_inv given.*

- void [fasp\\_smoother\\_dstr\\_gs\\_ascend](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv)

*Gauss-Seidel method as the smoother in the ascending manner.*

- void [fasp\\_smoother\\_dstr\\_gs\\_descend](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv)

*Gauss-Seidel method as the smoother in the descending manner.*

- void [fasp\\_smoother\\_dstr\\_gs\\_order](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [INT](#) \*mark)

*Gauss method as the smoother in the user-defined order.*

- void [fasp\\_smoother\\_dstr\\_gs\\_cf](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [INT](#) \*mark, const [INT](#) order)

*Gauss method as the smoother in the C-F manner.*

- void [fasp\\_smoother\\_dstr\\_sor](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, const [INT](#) order, [INT](#) \*mark, const [REAL](#) weight)

*SOR method as the smoother.*

- void [fasp\\_smoother\\_dstr\\_sor1](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, const [INT](#) order, [INT](#) \*mark, [REAL](#) \*diaginv, const [REAL](#) weight)

*SOR method as the smoother.*

- void [fasp\\_smoother\\_dstr\\_sor\\_ascend](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [REAL](#) weight)

*SOR method as the smoother in the ascending manner.*

- void [fasp\\_smoother\\_dstr\\_sor\\_descend](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [REAL](#) weight)

*SOR method as the smoother in the descending manner.*

- void [fasp\\_smoother\\_dstr\\_sor\\_order](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [INT](#) \*mark, [REAL](#) weight)

*SOR method as the smoother in the user-defined order.*

- void [fasp\\_smoother\\_dstr\\_sor\\_cf](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [REAL](#) \*diaginv, [INT](#) \*mark, const [INT](#) order, const [REAL](#) weight)

*SOR method as the smoother in the C-F manner.*

- void [fasp\\_generate\\_diaginv\\_block](#) ([dSTRmat](#) \*A, [ivector](#) \*neigh, [dvector](#) \*diaginv, [ivector](#) \*pivot)

*Generate inverse of diagonal block for block smoothers.*

- void [fasp\\_smoother\\_dstr\\_swz](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [dvector](#) \*diaginv, [ivector](#) \*pivot, [ivector](#) \*neigh, [ivector](#) \*order)

### 9.49.1 Detailed Description

Smoothers for [dSTRmat](#) matrices.

Note

This file contains Level-2 (ltr) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaSmallMat.c](#), [BlaSmallMatInv.c](#), [BlaSmallMatLU.c](#), and [BlaSpmvSTR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.49.2 Function Documentation

**9.49.2.1 fasp\_generate\_diaginv\_block()**

```
void fasp_generate_diaginv_block (
    dSTRmat * A,
    ivector * neigh,
    dvector * diaginv,
    ivector * pivot )
```

Generate inverse of diagonal block for block smoothers.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>neigh</i>	Pointer to ivector: neighborhoods
<i>diaginv</i>	Pointer to dvector: the inverse of the diagonals
<i>pivot</i>	Pointer to ivector: the pivot of diagonal blocks

**Author**

Xiaozhe Hu

**Date**

10/01/2011

Definition at line 1543 of file ltrSmootherSTR.c.

**9.49.2.2 fasp\_smoother\_dstr\_gs()**

```
void fasp_smoother_dstr_gs (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    const INT order,
    INT * mark )
```

Gauss-Seidel method as the smoother.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending manner DESCEND 21: in descending manner If mark != NULL USERDEFINED 0 : in the user-defined manner CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points
<i>mark</i>	Pointer to the user-defined ordering(when order=0) or CF_marker array(when order!=0)

**Author**

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 217 of file ltrSmootherSTR.c.

**9.49.2.3 fasp\_smoother\_dstr\_gs1()**

```
void fasp_smoother_dstr_gs1 (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    const INT order,
    INT * mark,
    REAL * diaginv )
```

Gauss-Seidel method as the smoother with diag\_inv given.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending manner DESCEND 21: in descending manner If mark != NULL USERDEFINED 0 : in the user-defined manner CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points
<i>mark</i>	Pointer to the user-defined ordering(when order=0) or CF_marker array(when order!=0)
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 277 of file ltrSmootherSTR.c.

**9.49.2.4 fasp\_smoother\_dstr\_gs\_ascend()**

```
void fasp_smoother_dstr_gs_ascend (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv )
```

Gauss-Seidel method as the smoother in the ascending manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 322 of file ltrSmootherSTR.c.

**9.49.2.5 fasp\_smoother\_dstr\_gs\_cf()**

```
void fasp_smoother_dstr_gs_cf (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    INT * mark,
    const INT order )
```

Gauss method as the smoother in the C-F manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>mark</i>	Pointer to the user-defined order array
<i>order</i>	Flag to indicate the order for smoothing CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 680 of file ltrSmootherSTR.c.

**9.49.2.6 fasp\_smoother\_dstr\_gs\_descend()**

```
void fasp_smoother_dstr_gs_descend (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv )
```

Gauss-Seidel method as the smoother in the descending manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
----------	---

## Parameters

<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 438 of file ltrSmootherSTR.c.

**9.49.2.7 fasp\_smoother\_dstr\_gs\_order()**

```
void fasp_smoother_dstr_gs_order (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    INT * mark )
```

Gauss method as the smoother in the user-defined order.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>mark</i>	Pointer to the user-defined order array

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 556 of file ltrSmootherSTR.c.

**9.49.2.8 fasp\_smoother\_dstr\_jacobi()**

```
void fasp_smoother_dstr_jacobi (
    dSTRmat * A,
    dvector * b,
    dvector * u )
```

Jacobi method as the smoother.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 43 of file ltrSmootherSTR.c.

**9.49.2.9 fasp\_smoother\_dstr\_jacobi1()**

```
void fasp_smoother_dstr_jacobi1 (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv )
```

Jacobi method as the smoother with diag\_inv given.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 92 of file ltrSmootherSTR.c.

**9.49.2.10 fasp\_smoother\_dstr\_sor()**

```
void fasp_smoother_dstr_sor (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    const INT order,
    INT * mark,
    const REAL weight )
```

SOR method as the smoother.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending manner DESCEND 21: in descending manner If mark != NULL USERDEFINED 0 : in the user-defined manner CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points
<i>mark</i>	Pointer to the user-defined ordering(when order=0) or CF_marker array(when order!=0)
<i>weight</i>	Over-relaxation weight

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 873 of file ltrSmootherSTR.c.

**9.49.2.11 fasp\_smoother\_dstr\_sor1()**

```
void fasp_smoother_dstr_sor1 (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    const INT order,
    INT * mark,
    REAL * diaginv,
    const REAL weight )
```

SOR method as the smoother.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending manner DESCEND 21: in descending manner If mark != NULL USERDEFINED 0 : in the user-defined manner CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points
<i>mark</i>	Pointer to the user-defined ordering(when order=0) or CF_marker array(when order!=0)
<i>diaginv</i>	Inverse of the diagonal entries
<i>weight</i>	Over-relaxation weight

## Author

Shiquan Zhang, Zhiyang Zhou



## Date

10/10/2010

Definition at line 935 of file ltrSmootherSTR.c.

**9.49.2.12 fasp\_smoother\_dstr\_sor\_ascend()**

```
void fasp_smoother_dstr_sor_ascend (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    REAL weight )
```

SOR method as the smoother in the ascending manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>weight</i>	Over-relaxation weight

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 981 of file ltrSmootherSTR.c.

**9.49.2.13 fasp\_smoother\_dstr\_sor\_cf()**

```
void fasp_smoother_dstr_sor_cf (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginv,
    INT * mark,
    const INT order,
    const REAL weight )
```

SOR method as the smoother in the C-F manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Parameters

<i>mark</i>	Pointer to the user-defined order array
<i>order</i>	Flag to indicate the order for smoothing CPFIRST 1 : C-points first and then F-points FPFIRST -1 : F-points first and then C-points
<i>weight</i>	Over-relaxation weight

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 1355 of file ltrSmootherSTR.c.

**9.49.2.14 fasp\_smoother\_dstr\_sor\_descend()**

```
void fasp_smoother_dstr_sor_descend (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    REAL * diagin,
    REAL weight )
```

SOR method as the smoother in the descending manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>weight</i>	Over-relaxation weight

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 1102 of file ltrSmootherSTR.c.

**9.49.2.15 fasp\_smoother\_dstr\_sor\_order()**

```
void fasp_smoother_dstr_sor_order (
    dSTRmat * A,
    dvector * b,
```

```

dvector * u,
REAL * diaginv,
INT * mark,
REAL weight )

```

SOR method as the smoother in the user-defined order.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>mark</i>	Pointer to the user-defined order array
<i>weight</i>	Over-relaxation weight

#### Author

Shiquan Zhang, Zhiyang Zhou

#### Date

10/10/2010

Definition at line 1224 of file ltrSmootherSTR.c.

## 9.50 KryPbcgs.c File Reference

Krylov subspace methods – Preconditioned BiCGstab.

```

#include <math.h>
#include <float.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"

```

### Functions

- [INT fasp\\_solver\\_dcsr\\_pbcgs](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned BiCGstab method for solving  $Au=b$  for CSR matrix.*
- [INT fasp\\_solver\\_dbsr\\_pbcgs](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned BiCGstab method for solving  $Au=b$  for BSR matrix.*
- [INT fasp\\_solver\\_dblc\\_pbcgs](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned BiCGstab method for solving  $Au=b$  for BLC matrix.*
- [INT fasp\\_solver\\_dstr\\_pbcgs](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned BiCGstab method for solving  $Au=b$  for STR matrix.*
- [INT fasp\\_solver\\_pbcgs](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*

### 9.50.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

This version is based on Matlab 2011a – Chunsheng Feng

See [KrySPbcgs.c](#) for a safer version

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2016–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.50.2 Function Documentation

#### 9.50.2.1 fasp\_solver\_dblc\_pbcgs()

```
INT fasp_solver_dblc_pbcgs (
    dBLMat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  for BLC matrix.

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>u</i>	Pointer to dvector of DOFs
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chunsheng Feng

## Date

03/04/2016

Definition at line 715 of file KryPbcgs.c.

**9.50.2.2 fasp\_solver\_dbsr\_pbcgs()**

```

INT fasp_solver_dbsr_pbcgs (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned BiCGstab method for solving  $Au=b$  for BSR matrix.

## Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>u</i>	Pointer to dvector of DOFs
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chunsheng Feng

## Date

03/04/2016

Definition at line 388 of file KryPbcgs.c.

**9.50.2.3 fasp\_solver\_dcsr\_pbcgs()**

```

INT fasp_solver_dcsr_pbcgs (
    dCSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,

```

```
const SHORT StopType,
const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  for CSR matrix.

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>u</i>	Pointer to dvector of DOFs
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chunsheng Feng

#### Date

03/04/2016

Definition at line 62 of file KryPbcgs.c.

#### 9.50.2.4 fasp\_solver\_dstr\_pbcgs()

```
INT fasp_solver_dstr_pbcgs (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  for STR matrix.

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>u</i>	Pointer to dvector of DOFs
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chunsheng Feng

**Date**

03/04/2016

Definition at line 1042 of file KryPbcgs.c.

**9.50.2.5 fasp\_solver\_pbcgs()**

```

INT fasp_solver_pbcgs (
    mxv_matfree * mf,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ .

**Parameters**

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : spmv operation
<i>b</i>	Pointer to dvector of right hand side
<i>u</i>	Pointer to dvector of DOFs
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chunsheng Feng

**Date**

03/04/2016

Definition at line 1369 of file KryPbcgs.c.

## 9.51 KryPcg.c File Reference

Krylov subspace methods – Preconditioned CG.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_pcg](#) (dCSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT StopType, const SHORT PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dbsr\\_pcg](#) (dBSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT StopType, const SHORT PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dblc\\_pcg](#) (dBLCmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT StopType, const SHORT PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dstr\\_pcg](#) (dSTRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT StopType, const SHORT PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_pcg](#) (mxv\_matfree \*mf, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT StopType, const SHORT PrtLvl)  
*Preconditioned conjugate gradient (CG) method for solving  $Au=b$ .*

### 9.51.1 Detailed Description

Krylov subspace methods – Preconditioned CG.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

See [KrySPcg.c](#) for a safer version

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2009–Present by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

Abstract algorithm

PCG method to solve  $Ax=b$  is to generate  $\{x_k\}$  to approximate  $x$

Step 0. Given  $A$ ,  $b$ ,  $x_0$ ,  $M$

Step 1. Compute residual  $r_0 = b - Ax_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1}r_0$ ,  $p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:MaxIt$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha p_k$ ;



- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha(A * p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;
- print the result of k-th iteration; END FOR

Convergence check:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check:

- IF  $\text{norm}(\alpha * p_k)/\text{norm}(x_{k+1}) < \text{tol\_stag}$ 
  1. compute  $r = b - A * x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Stag\_Check ) restart;
- END IF

Residual check:

- IF  $\text{norm}(r_{k+1})/\text{norm}(b) < \text{tol}$ 
  1. compute the real residual  $r = b - A * x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Res\_Check ) restart;
- END IF

## 9.51.2 Function Documentation

### 9.51.2.1 fasp\_solver\_dblc\_pcg()

```

INT fasp_solver_dblc_pcg (
    dBLMat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned conjugate gradient method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <b>dBLMat</b> : coefficient matrix
<i>b</i>	Pointer to <b>dvector</b> : right hand side
<i>u</i>	Pointer to <b>dvector</b> : unknowns
<i>pc</i>	Pointer to <b>precond</b> : structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

05/24/2010

Modified by Chensong Zhang on 03/28/2013

Definition at line 684 of file KryPcg.c.

**9.51.2.2 fasp\_solver\_dbsr\_pcg()**

```

INT fasp_solver_dbsr_pcg (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned conjugate gradient method for solving  $Au=b$ .

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

05/26/2014

Definition at line 390 of file KryPcg.c.

### 9.51.2.3 fasp\_solver\_dcsr\_pcg()

```

INT fasp_solver_dcsr_pcg (
    dCSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned conjugate gradient method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang, Xiaozhe Hu, Shiquan Zhang

#### Date

05/06/2010

Definition at line 98 of file KryPcg.c.

### 9.51.2.4 fasp\_solver\_dstr\_pcg()

```

INT fasp_solver_dstr_pcg (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned conjugate gradient method for solving  $Au=b$ .

## Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Zhiyang Zhou

## Date

04/25/2010

Modified by Chensong Zhang on 03/28/2013

Definition at line 978 of file KryPcg.c.

## 9.51.2.5 fasp\_solver\_pcg()

```

INT fasp_solver_pcg (
    mxv_matfree * mf,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned conjugate gradient (CG) method for solving  $Au=b$ .

## Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang, Xiaozhe Hu, Shiquan Zhang

**Date**

05/06/2010

Modified by Feiteng Huang on 09/19/2012: matrix free  
Definition at line 1272 of file KryPcg.c.

## 9.52 KryPgcg.c File Reference

Krylov subspace methods – Preconditioned generalized CG.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

**Functions**

- [INT fasp\\_solver\\_dcsr\\_pgcg](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned generalized conjugate gradient (GCG) method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_pgcg](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned generalized conjugate gradient (GCG) method for solving  $Au=b$ .*

### 9.52.1 Detailed Description

Krylov subspace methods – Preconditioned generalized CG.

**Note**

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), and [BlaSpmvCSR.c](#)

Reference: Concus, P. and Golub, G.H. and O'Leary, D.P. A Generalized Conjugate Gradient Method for the Numerical: Solution of Elliptic Partial Differential Equations, Computer Science Department, Stanford University, 1976  
Copyright (C) 2012–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.52.2 Function Documentation

### 9.52.2.1 fasp\_solver\_dcsr\_pgcg()

```

INT fasp_solver_dcsr_pgcg (
    dCSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned generalized conjugate gradient (GCG) method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xiaozhe Hu

#### Date

01/01/2012

Modified by Chensong Zhang on 05/01/2012

Definition at line 60 of file KryPgcg.c.

### 9.52.2.2 fasp\_solver\_pgcg()

```

INT fasp_solver_pgcg (
    mxv_matfree * mf,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned generalized conjugate gradient (GCG) method for solving  $Au=b$ .

## Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type – DOES not support this parameter
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

01/01/2012

## Note

Not completely implemented yet! –Chensong

Modified by Feiteng Huang on 09/26/2012: matrix free  
Definition at line 213 of file KryPgcr.c.

## 9.53 KryPgcr.c File Reference

Krylov subspace methods – Preconditioned GCR.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_pgcr](#) (dCSRmat \*A, dvector \*b, dvector \*x, precondition \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*A preconditioned GCR method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dblc\\_pgcr](#) (dBLCmat \*A, dvector \*b, dvector \*x, precondition \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*A preconditioned GCR method for solving  $Au=b$ .*

### 9.53.1 Detailed Description

Krylov subspace methods – Preconditioned GCR.

## Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvCSR.c](#), and [BlaVector.c](#)

Copyright (C) 2014–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

TODO: Use one single function for all! –Chensong

## 9.53.2 Function Documentation

### 9.53.2.1 fasp\_solver\_dblc\_pgcr()

```

INT fasp_solver_dblc_pgcr (
    dBLMat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned GCR method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>x</i>	Pointer to dvector of dofs
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stoppage
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restart number for GCR
<i>StopType</i>	Stopping type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

Reference: YVAN NOTAY "AN AGGREGATION-BASED ALGEBRAIC MULTIGRID METHOD"

#### Author

Zheng Li

#### Date

12/23/2014

Definition at line 249 of file KryPgcr.c.

### 9.53.2.2 fasp\_solver\_dcsr\_pgcr()

```

INT fasp_solver_dcsr_pgcr (
    dCSRmat * A,

```



```

dvector * b,
dvector * x,
precond * pc,
const REAL tol,
const INT MaxIt,
const SHORT restart,
const SHORT StopType,
const SHORT PrtLvl )

```

A preconditioned GCR method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to dvector of right hand side
<i>x</i>	Pointer to dvector of dofs
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopage
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restart number for GCR
<i>StopType</i>	Stopping type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

Reference: YVAN NOTAY "AN AGGREGATION-BASED ALGEBRAIC MULTIGRID METHOD"

#### Author

Zheng Li

#### Date

12/23/2014

Definition at line 55 of file KryPgcr.c.

## 9.54 KryPgmres.c File Reference

Krylov subspace methods – Right-preconditioned GMRes.

```

#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"

```

### Functions

- `INT fasp_solver_dcsr_pgmres` (dCSRmat \*A, dvector \*b, dvector \*x, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT StopType, const SHORT PrtLvl)  
*Right preconditioned GMRES method for solving  $Au=b$ .*
- `INT fasp_solver_dbsr_pgmres` (dBSRmat \*A, dvector \*b, dvector \*x, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT StopType, const SHORT PrtLvl)

*Preconditioned GMRES method for solving  $Au=b$ .*

- `INT fasp_solver_dblc_pgmres` (`dbLCmat` \*A, `dvector` \*b, `dvector` \*x, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` restart, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned GMRES method for solving  $Au=b$ .*

- `INT fasp_solver_dstr_pgmres` (`dSTRmat` \*A, `dvector` \*b, `dvector` \*x, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` restart, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned GMRES method for solving  $Au=b$ .*

- `INT fasp_solver_pgmres` (`mxv_matfree` \*mf, `dvector` \*b, `dvector` \*x, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` restart, const `SHORT` StopType, const `SHORT` PrtLvl)

*Solve " $Ax=b$ " using PGMRES (right preconditioned) iterative method.*

## 9.54.1 Detailed Description

Krylov subspace methods – Right-preconditioned GMRes.

Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

See also [KryPvgmres.c](#) for a variable restarting version.

See [KrySPgmres.c](#) for a safer version

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2010–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

## 9.54.2 Function Documentation

### 9.54.2.1 `fasp_solver_dblc_pgmres()`

```
INT fasp_solver_dblc_pgmres (
    dbLCmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned GMRES method for solving  $Au=b$ .

Parameters

<i>A</i>	Pointer to <code>dbLCmat</code> : coefficient matrix
<i>b</i>	Pointer to <code>dvector</code> : right hand side
<i>x</i>	Pointer to <code>dvector</code> : unknowns
<i>pc</i>	Pointer to <code>precond</code> : structure of precondition
<i>tol</i>	Tolerance for stopping

## Parameters

<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

05/24/2010

Modified by Chensong Zhang on 04/05/2013: add StopType and safe check  
Definition at line 675 of file KryPgmres.c.

## 9.54.2.2 fasp\_solver\_dbsr\_pgmres()

```

INT fasp_solver_dbsr_pgmres (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned GMRES method for solving  $Au=b$ .

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou

**Date**

2010/12/21

Modified by Chensong Zhang on 04/05/2013: add StopType and safe check  
Definition at line 370 of file KryPgmres.c.

**9.54.2.3 fasp\_solver\_dcsr\_pgmres()**

```

INT fasp_solver_dcsr_pgmres (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Right preconditioned GMRES method for solving  $Au=b$ .

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou

## Date

2010/11/28

Modified by Chensong Zhang on 04/05/2013: Add StopType and safe check Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate Modified by Chensong Zhang on 09/21/2014: Add comments and reorganize code

Definition at line 67 of file KryPgmres.c.

**9.54.2.4 fasp\_solver\_dstr\_pgmres()**

```

INT fasp_solver_dstr_pgmres (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned GMRES method for solving  $Au=b$ .

## Parameters

<i>A</i>	Pointer to <b>dSTRmat</b> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Zhiyang Zhou

## Date

2010/11/28

Modified by Chensong Zhang on 04/05/2013: add StopType and safe check  
Definition at line 979 of file KryPgmres.c.

**9.54.2.5 fasp\_solver\_pgmres()**

```

INT fasp_solver_pgmres (
    mxv_matfree * mf,

```

```

dvector * b,
dvector * x,
precond * pc,
const REAL tol,
const INT MaxIt,
const SHORT restart,
const SHORT StopType,
const SHORT PrtLvl )

```

Solve "Ax=b" using PGMRES (right preconditioned) iterative method.

#### Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type – DOES not support this parameter
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Zhiyang Zhou

#### Date

2010/11/28

Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate  
Definition at line 1283 of file KryPgmres.c.

## 9.55 KryPminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual.

```

#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"

```

### Functions

- [INT fasp\\_solver\\_dcsr\\_pminres](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)

*A preconditioned minimal residual (Minres) method for solving  $Au=b$ .*

- `INT fasp_solver_dblc_pminres` (`dBLCmat *A`, `dvector *b`, `dvector *u`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT StopType`, `const SHORT PrtLvl`)  
A preconditioned minimal residual (Minres) method for solving  $Au=b$ .
- `INT fasp_solver_dstr_pminres` (`dSTRmat *A`, `dvector *b`, `dvector *u`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT StopType`, `const SHORT PrtLvl`)  
A preconditioned minimal residual (Minres) method for solving  $Au=b$ .
- `INT fasp_solver_pminres` (`mxv_matfree *mf`, `dvector *b`, `dvector *u`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT StopType`, `const SHORT PrtLvl`)  
A preconditioned minimal residual (Minres) method for solving  $Au=b$ .

### 9.55.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c.o](#)

See [KrySPminres.c](#) for a safer version

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2012–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

TODO: Use one single function for all! –Chensong

### 9.55.2 Function Documentation

#### 9.55.2.1 fasp\_solver\_dblc\_pminres()

```
INT fasp_solver_dblc_pminres (
    dBLCmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

A preconditioned minimal residual (Minres) method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <code>dBLCmat</code> : coefficient matrix
<i>b</i>	Pointer to <code>dvector</code> : right hand side
<i>u</i>	Pointer to <code>dvector</code> : unknowns
<i>pc</i>	Pointer to <code>precond</code> : structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

05/01/2012

Rewritten based on the original version by Xiaozhe Hu 05/24/2010 Modified by Chensong Zhang on 04/09/2013  
Definition at line 475 of file KryPminres.c.

**9.55.2.2 fasp\_solver\_dcsr\_pminres()**

```

INT fasp_solver_dcsr_pminres (
    dCSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned minimal residual (Minres) method for solving  $Au=b$ .

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

05/01/2012

Rewritten based on the original version by Shiquan Zhang 05/10/2010 Modified by Chensong Zhang on 04/09/2013  
Definition at line 62 of file KryPminres.c.



### 9.55.2.3 fasp\_solver\_dstr\_pminres()

```

INT fasp_solver_dstr_pminres (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned minimal residual (Minres) method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/09/2013

Definition at line 885 of file KryPminres.c.

### 9.55.2.4 fasp\_solver\_pminres()

```

INT fasp_solver_pminres (
    mxv_matfree * mf,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned minimal residual (Minres) method for solving  $Au=b$ .

## Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Shiquan Zhang

## Date

10/24/2010

Rewritten by Chensong Zhang on 05/01/2012

Definition at line 1296 of file KryPminres.c.

## 9.56 KryPvfgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restarting FGMRes.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_pvfgmres](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.*
- [INT fasp\\_solver\\_dbsr\\_pvfgmres](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.*
- [INT fasp\\_solver\\_dblc\\_pvfgmres](#) ([dBLMat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PFGMRES (right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.*
- [INT fasp\\_solver\\_pvfgmres](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.*

### 9.56.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restarting FGMRes.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), and [BlaSpmvCSR.c](#)

This file is modified from [KryPvfgmres.c](#)

Reference: A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMRES(m) Journal of Computational and Applied Mathematics, 230 (2009) pp. 751-761. UCRL-JRNL-235266.  
Copyright (C) 2012–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.56.2 Function Documentation

#### 9.56.2.1 fasp\_solver\_dblc\_pvfgmres()

```
INT fasp_solver_dblc_pvfgmres (
    dBLMat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Solve "Ax=b" using PFGMRES (right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.

#### Parameters

<i>A</i>	Pointer to coefficient matrix
<i>b</i>	Pointer to right hand side vector
<i>x</i>	Pointer to solution vector
<i>MaxIt</i>	Maximal iteration number allowed
<i>tol</i>	Tolerance
<i>pc</i>	Pointer to preconditioner data
<i>PrtLvl</i>	How much information to print out
<i>StopType</i>	Stopping criterion, i.e. $\ r_k\ /\ r_0\  < tol$
<i>restart</i>	Number of restart for GMRES

#### Returns

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

01/04/2012

**Note**

Based on Zhiyang Zhou's pvgmres.c

Modified by Chunsheng Feng on 07/22/2013: Add adaptive memory allocate  
 Modified by Chensong Zhang on 05/09/2015: Clean up for stopping types  
 Definition at line 714 of file KryPvfgmres.c.

**9.56.2.2 fasp\_solver\_dbsr\_pvfgmres()**

```

INT fasp_solver_dbsr_pvfgmres (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Solve " $Ax=b$ " using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type – DO not support this parameter
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

## Date

02/05/2012

Modified by Chunsheng Feng on 07/22/2013: Add adaptive memory allocate Modified by Chensong Zhang on 05/09/2015: Clean up for stopping types  
Definition at line 389 of file KryPvfgmres.c.

## 9.56.2.3 fasp\_solver\_dcsr\_pvfgmres()

```

INT fasp_solver_dcsr_pvfgmres (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.

## Parameters

<i>A</i>	Pointer to <b>dCSRmat</b> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type – DO not support this parameter
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

01/04/2012

Modified by Chunsheng Feng on 07/22/2013: Add adaptive memory allocate Modified by Chensong Zhang on 05/09/2015: Clean up for stopping types  
Definition at line 67 of file KryPvfgmres.c.

#### 9.56.2.4 fasp\_solver\_pvfgmres()

```

INT fasp_solver_pvfgmres (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration and flexible preconditioner can be used.

##### Parameters

<i>mf</i>	Pointer to <code>mxv_matfree</code> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type – DO not support this parameter
<i>PrtLvl</i>	How much information to print out

##### Returns

Iteration number if converges; ERROR otherwise.

##### Author

Xiaozhe Hu

##### Date

01/04/2012

Modified by Feiteng Huang on 09/26/2012: matrix free Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate

Definition at line 1036 of file KryPvfgmres.c.

## 9.57 KryPvfgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRes.

```

#include <math.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "KryUtil.inl"

```

## Functions

- `INT fasp_solver_dcsr_pvgmres` (`dCSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT StopType`, `const SHORT PrtLvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dbsr_pvgmres` (`dBSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT StopType`, `const SHORT PrtLvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dblc_pvgmres` (`dBLCmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT StopType`, `const SHORT PrtLvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dstr_pvgmres` (`dSTRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT StopType`, `const SHORT PrtLvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_pvgmres` (`mxv_matfree *mf`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `SHORT restart`, `const SHORT StopType`, `const SHORT PrtLvl`)  
*Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.*

### 9.57.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restart GMRes.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

See [KrySPvgmres.c](#) for a safer version

Reference: A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMRES(m) *Journal of Computational and Applied Mathematics*, 230 (2009) pp. 751-761. UCRL-JRNL-235266.  
 Copyright (C) 2010–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.57.2 Function Documentation

#### 9.57.2.1 `fasp_solver_dblc_pvgmres()`

```
INT fasp_solver_dblc_pvgmres (
    dBLCmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

04/05/2013

Definition at line 757 of file KryPvgmres.c.

## 9.57.2.2 fasp\_solver\_dbsr\_pvgmres()

```

INT fasp_solver_dbsr_pvgmres (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out



**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou

**Date**

12/21/2011

Modified by Chensong Zhang on 04/06/2013: Add stop type support  
Definition at line 413 of file KryPvgmres.c.

**9.57.2.3 fasp\_solver\_dcsr\_pvgmres()**

```

INT fasp_solver_dcsr_pvgmres (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou

## Date

2010/12/14

Modified by Chensong Zhang on 04/06/2013: Add stop type support  
 Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate

Definition at line 66 of file KryPvbmres.c.

#### 9.57.2.4 fasp\_solver\_dstr\_pvgmres()

```
INT fasp_solver_dstr_pvgmres (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Zhiyang Zhou

## Date

2010/12/14

Modified by Chensong Zhang on 04/06/2013: Add stop type support  
 Definition at line 1104 of file KryPvbmres.c.

#### 9.57.2.5 fasp\_solver\_pvgmres()

```
INT fasp_solver_pvgmres (
    mxv_matfree * mf,
```

```

dvector * b,
dvector * x,
precond * pc,
const REAL tol,
const INT MaxIt,
SHORT restart,
const SHORT StopType,
const SHORT PrtLvl )

```

Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.

#### Parameters

<i>mf</i>	Pointer to <code>mxv_matfree</code> : spmv operation
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type – DOES not support this parameter
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Zhiyang Zhou

#### Date

2010/12/14

Modified by Feiteng Huang on 09/26/2012: matrix free Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate

Definition at line 1451 of file KryPvgmres.c.

## 9.58 KrySPbcgs.c File Reference

Krylov subspace methods – Preconditioned BiCGstab with safety net.

```

#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"

```

#### Functions

- `INT fasp_solver_dcsr_spbcgs` (const `dCSRmat` \*A, const `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned BiCGstab method for solving  $Au=b$  with safety net.*

- `INT fasp_solver_dbsr_spgbgs` (const `DBSRmat` \*A, const `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned BiCGstab method for solving  $Au=b$  with safety net.*

- `INT fasp_solver_dblc_spgbgs` (const `DBLCmat` \*A, const `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned BiCGstab method for solving  $Au=b$  with safety net.*

- `INT fasp_solver_dstr_spgbgs` (const `dSTRmat` \*A, const `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` StopType, const `SHORT` PrtLvl)

*Preconditioned BiCGstab method for solving  $Au=b$  with safety net.*

## 9.58.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab with safety net.

Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

The ‘best’ iterative solution will be saved and used upon exit; See [KryPbcgs.c](#) for a version without safety net

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

TODO: Update this version with the new BiCGstab implementation! –Chensong  
TODO: Use one single function for all! –Chensong

## 9.58.2 Function Documentation

### 9.58.2.1 fasp\_solver\_dblc\_spgbgs()

```
INT fasp_solver_dblc_spgbgs (
    const DBLCmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  with safety net.

Parameters

<i>A</i>	Pointer to <code>DBLCmat</code> : the coefficient matrix
<i>b</i>	Pointer to <code>dvector</code> : the right hand side
<i>u</i>	Pointer to <code>dvector</code> : the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations

## Parameters

<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

03/31/2013

Definition at line 843 of file KrySPbcgs.c.

## 9.58.2.2 fasp\_solver\_dbsr\_spbcgs()

```

INT fasp_solver_dbsr_spbcgs (
    const dBSRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned BiCGstab method for solving  $Au=b$  with safety net.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

Date

03/31/2013

Definition at line 452 of file KrySPbcgs.c.

### 9.58.2.3 fasp\_solver\_dcsr\_spgcgs()

```
INT fasp_solver_dcsr_spgcgs (
    const dCSRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  with safety net.

Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

Returns

Iteration number if converges; ERROR otherwise.

Author

Chensong Zhang

Date

03/31/2013

Definition at line 61 of file KrySPbcgs.c.

### 9.58.2.4 fasp\_solver\_dstr\_spgcgs()

```
INT fasp_solver_dstr_spgcgs (
    const dSTRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
```

```
const SHORT StopType,
const SHORT PrtLvl )
```

Preconditioned BiCGstab method for solving  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

03/31/2013

Definition at line 1234 of file KrySPbcgs.c.

## 9.59 KrySPcg.c File Reference

Krylov subspace methods – Preconditioned CG with safety net.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_spcg](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_dblc\\_spcg](#) (const [dBLCmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_dstr\\_spcg](#) (const [dSTRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.*

### 9.59.1 Detailed Description

Krylov subspace methods – Preconditioned CG with safety net.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvCSR.c](#), [BlaSpmvSTR.c](#), and [BlaVector.c](#)

The 'best' iterative solution will be saved and used upon exit; See [KryPcg.c](#) for a version without safety net

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.59.2 Function Documentation

#### 9.59.2.1 fasp\_solver\_dblc\_spcg()

```
INT fasp_solver_dblc_spcg (
    const dBLMat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBLMat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang



Date

03/28/2013

Definition at line 393 of file KrySPcg.c.

### 9.59.2.2 fasp\_solver\_dcsr\_spcg()

```
INT fasp_solver_dcsr_spcg (
    const dCSRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

Date

03/28/2013

Definition at line 60 of file KrySPcg.c.

### 9.59.2.3 fasp\_solver\_dstr\_spcg()

```
INT fasp_solver_dstr_spcg (
    const dSTRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
```

```
const SHORT StopType,
const SHORT PrtLvl )
```

Preconditioned conjugate gradient method for solving  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>MaxIt</i>	Maximal number of iterations
<i>tol</i>	Tolerance for stopping
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

03/28/2013

Definition at line 726 of file KrySPcg.c.

## 9.60 KrySPgmres.c File Reference

Krylov subspace methods – Preconditioned GMRes with safety net.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_spgmres](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- [INT fasp\\_solver\\_dbsr\\_spgmres](#) (const [dBSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- [INT fasp\\_solver\\_dblc\\_spgmres](#) (const [dBLCmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- [INT fasp\\_solver\\_dstr\\_spgmres](#) (const [dSTRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*

## 9.60.1 Detailed Description

Krylov subspace methods – Preconditioned GMRes with safety net.

### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

See also `pgmres.c` for a variable restarting version.

The ‘best’ iterative solution will be saved and used upon exit; See [KryPgmres.c](#) for a version without safety net

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

## 9.60.2 Function Documentation

### 9.60.2.1 fasp\_solver\_dblc\_spgmres()

```
INT fasp_solver_dblc_spgmres (
    const dBLCmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

#### Parameters

<i>A</i>	Pointer to <a href="#">dBLCmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

04/05/2013

Definition at line 752 of file KrySPgmres.c.

**9.60.2.2 fasp\_solver\_dbsr\_spgmres()**

```

INT fasp_solver_dbsr_spgmres (
    const dBSRmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

04/05/2013

Definition at line 409 of file KrySPgmres.c.

**9.60.2.3 fasp\_solver\_dcsr\_spgmres()**

```

INT fasp_solver_dcsr_spgmres (
    const dCSRmat * A,

```

```

const dvector * b,
dvector * x,
precond * pc,
const REAL tol,
const INT MaxIt,
SHORT restart,
const SHORT StopType,
const SHORT PrtLvl )

```

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/05/2013

Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate  
Definition at line 66 of file KrySPgmres.c.

#### 9.60.2.4 fasp\_solver\_dstr\_spgmres()

```

INT fasp_solver_dstr_spgmres (
    const dSTRmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

## Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

04/05/2013

Definition at line 1095 of file KrySPgmres.c.

## 9.61 KrySPminres.c File Reference

Krylov subspace methods – Preconditioned MINRES with safety net.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_spmminres](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_dblc\\_spmminres](#) (const [dBLCmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_dstr\\_spmminres](#) (const [dSTRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*

### 9.61.1 Detailed Description

Krylov subspace methods – Preconditioned MINRES with safety net.

**Note**

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

The 'best' iterative solution will be saved and used upon exit; See [KryPminres.c](#) for a version without safety net

Reference: Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

**9.61.2 Function Documentation****9.61.2.1 fasp\_solver\_dblc\_spmminres()**

```
INT fasp_solver_dblc_spmminres (
    const dBLCmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )
```

A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBLCmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

04/09/2013

Definition at line 511 of file KrySPminres.c.

### 9.61.2.2 fasp\_solver\_dcsr\_spmminres()

```

INT fasp_solver_dcsr_spmminres (
    const dCSRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/09/2013

Definition at line 60 of file KrySPminres.c.

### 9.61.2.3 fasp\_solver\_dstr\_spmminres()

```

INT fasp_solver_dstr_spmminres (
    const dSTRmat * A,
    const dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT StopType,
    const SHORT PrtLvl )

```

A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.



## Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>u</i>	Pointer to dvector: unknowns
<i>MaxIt</i>	Maximal number of iterations
<i>tol</i>	Tolerance for stopping
<i>pc</i>	Pointer to structure of precondition (precond)
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

04/09/2013

Definition at line 962 of file KrySPminres.c.

## 9.62 KrySPvgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRes with safety net.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_spvgmres](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.*
- [INT fasp\\_solver\\_dbsr\\_spvgmres](#) (const [dBSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.*
- [INT fasp\\_solver\\_dblc\\_spvgmres](#) (const [dBLCmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Preconditioned GMRES method for solving Au=b.*
- [INT fasp\\_solver\\_dstr\\_spvgmres](#) (const [dSTRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) StopType, const [SHORT](#) PrtLvl)  
*Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.*

### 9.62.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restart GMRes with safety net.

#### Note

This file contains Level-3 (Kry) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [BlaSpmvSTR.c](#)

The 'best' iterative solution will be saved and used upon exit; See [KryPvgmres.c](#) a version without safety net

Reference: A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMRES(m) Journal of Computational and Applied Mathematics, 230 (2009) pp. 751-761. UCRL-JRNL-235266.  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Use one single function for all! –Chensong

### 9.62.2 Function Documentation

#### 9.62.2.1 fasp\_solver\_dblc\_spgmres()

```
INT fasp_solver_dblc_spgmres (
    const dBLMat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Preconditioned GMRES method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <a href="#">dBLMat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

04/06/2013

Definition at line 829 of file KrySPvgmres.c.

**9.62.2.2 fasp\_solver\_dbsr\_spvgmres()**

```
INT fasp_solver_dbsr_spvgmres (
    const dBSRmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )
```

Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.

## Parameters

<i>A</i>	Pointer to dBSRmat: coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

04/06/2013

Definition at line 449 of file KrySPvgmres.c.

### 9.62.2.3 fasp\_solver\_dcsr\_spvgmres()

```

INT fasp_solver_dcsr_spvgmres (
    const dCSRmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT StopType,
    const SHORT PrtLvl )

```

Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/06/2013

Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate  
Definition at line 68 of file KrySPvgmres.c.

### 9.62.2.4 fasp\_solver\_dstr\_spvgmres()

```

INT fasp_solver_dstr_spvgmres (
    const dSTRmat * A,
    const dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,

```

```
const SHORT StopType,
const SHORT PrtLvl )
```

Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.

#### Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : coefficient matrix
<i>b</i>	Pointer to dvector: right hand side
<i>x</i>	Pointer to dvector: unknowns
<i>pc</i>	Pointer to structure of precondition (precond)
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>StopType</i>	Stopping criteria type
<i>PrtLvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/06/2013

Definition at line 1210 of file KrySPvgmres.c.

## 9.63 PreAMGCoarsenCR.c File Reference

Coarsening with Brannick-Falgout strategy.

```
#include <math.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "PreAMGUtil.inl"
```

### Functions

- [INT fasp\\_amg\\_coarsening\\_cr](#) (const [INT](#) i\_0, const [INT](#) i\_n, [dCSRmat](#) \*A, [ivector](#) \*vertices, [AMG\\_param](#) \*param)  
*CR coarsening.*

#### 9.63.1 Detailed Description

Coarsening with Brannick-Falgout strategy.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxMemory.c](#), [AuxThreads.c](#), and [ltrSmootherCSRcr.c](#)

Copyright (C) 2010–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

// TODO: Not completed! —Chensong

## 9.63.2 Function Documentation

### 9.63.2.1 fasp\_amg\_coarsening\_cr()

```
INT fasp_amg_coarsening_cr (
    const INT i_0,
    const INT i_n,
    dCSRmat * A,
    ivector * vertices,
    AMG_param * param )
```

CR coarsening.

#### Parameters

<i>i_0</i>	Starting index
<i>i_n</i>	Ending index
<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix (index starts from 0)
<i>vertices</i>	Pointer to CF, 0: Fpt (current level) or 1: Cpt
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

#### Returns

Number of coarse level points

#### Author

James Brannick

#### Date

04/21/2010

#### Note

vertices = 0: fine; 1: coarse; 2: isolated or special

Modified by Chunsheng Feng, Zheng Li on 10/14/2012 CR STAGES  
Definition at line 62 of file PreAMGCoarsenCR.c.

## 9.64 PreAMGCoarsenRS.c File Reference

Coarsening with a modified Ruge-Stuben strategy.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "PreAMGUtil.inl"
```

## Functions

- [SHORT fasp\\_amg\\_coarsening\\_rs](#) ([dCSRmat](#) \*A, [ivector](#) \*vertices, [dCSRmat](#) \*P, [iCSRmat](#) \*S, [AMG\\_param](#) \*param)

*Standard and aggressive coarsening schemes.*

### 9.64.1 Detailed Description

Coarsening with a modified Ruge-Stuben strategy.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxThreads.c](#), [AuxVector.c](#), [BlaSparseCSR.c](#), and [PreAMGCoarsenCR.c](#)

Reference: Multigrid by U. Trottenberg, C. W. Oosterlee and A. Schuller Appendix P475 A.7 (by A. Brandt, P. Oswald and K. Stuben) Academic Press Inc., San Diego, CA, 2001.  
Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.64.2 Function Documentation

#### 9.64.2.1 fasp\_amg\_coarsening\_rs()

```
SHORT fasp_amg_coarsening_rs (
    dCSRmat * A,
    ivector * vertices,
    dCSRmat * P,
    iCSRmat * S,
    AMG_param * param )
```

Standard and aggressive coarsening schemes.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : Coefficient matrix (index starts from 0)
<i>vertices</i>	Indicator vector for the C/F splitting of the variables
<i>P</i>	Interpolation matrix (nonzero pattern only)
<i>S</i>	Strong connection matrix
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Xuehai Huang, Chensong Zhang, Xiaozhe Hu, Ludmil Zikatanov

#### Date

09/06/2010

**Note**

vertices = 0: fine; 1: coarse; 2: isolated or special

Modified by Xiaozhe Hu on 05/23/2011: add strength matrix as an argument Modified by Xiaozhe Hu on 04/24/2013: modify aggressive coarsening Modified by Chensong Zhang on 04/28/2013: remove linked list Modified by Chensong Zhang on 05/11/2013: restructure the code  
Definition at line 73 of file PreAMGCoarsenRS.c.

## 9.65 PreAMGInterp.c File Reference

Direct and standard interpolations for classical AMG.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_amg\\_interp](#) ([dCSRmat](#) \*A, [ivector](#) \*vertices, [dCSRmat](#) \*P, [iCSRmat](#) \*S, [AMG\\_param](#) \*param)

*Generate interpolation operator P.*

#### 9.65.1 Detailed Description

Direct and standard interpolations for classical AMG.

**Note**

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxThreads.c](#), and [PreAMGInterpEM.c](#)

Reference: U. Trottenberg, C. W. Oosterlee, and A. Schuller Multigrid (Appendix A: An Intro to Algebraic Multigrid) Academic Press Inc., San Diego, CA, 2001 With contributions by A. Brandt, P. Oswald and K. Stuben.  
Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

#### 9.65.2 Function Documentation

##### 9.65.2.1 fasp\_amg\_interp()

```
void fasp_amg_interp (
    dCSRmat * A,
    ivector * vertices,
    dCSRmat * P,
    iCSRmat * S,
    AMG\_param * param )
```

Generate interpolation operator P.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> coefficient matrix (index starts from 0)
<i>vertices</i>	Indicator vector for the C/F splitting of the variables



## Parameters

<i>P</i>	Prolongation (input: nonzero pattern, output: prolongation)
<i>S</i>	Strong connection matrix
<i>param</i>	AMG parameters

## Author

Xuehai Huang, Chensong Zhang

## Date

04/04/2010

Modified by Xiaozhe Hu on 05/23/2012: add S as input Modified by Chensong Zhang on 09/12/2012: clean up and debug interp\_RS Modified by Chensong Zhang on 05/14/2013: reconstruct the code  
Definition at line 63 of file PreAMGInterp.c.

## 9.66 PreAMGInterpEM.c File Reference

Interpolation operators for AMG based on energy-min.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_amg\\_interp\\_em](#) ([dCSRmat](#) \*A, [ivector](#) \*vertices, [dCSRmat](#) \*P, [AMG\\_param](#) \*param)  
*Energy-min interpolation.*

#### 9.66.1 Detailed Description

Interpolation operators for AMG based on energy-min.

## Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxThreads.c](#), [AuxVector.c](#), [BlaSmallMatLU.c](#), [BlaSparseCSR.c](#), [KryPcg.c](#), and [PreCSR.c](#)

Reference: J. Xu and L. Zikatanov On An Energy Minimizing Basis in Algebraic Multigrid Methods, Computing and visualization in sciences, 2003

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.66.2 Function Documentation

### 9.66.2.1 fasp\_amg\_interp\_em()

```
void fasp_amg_interp_em (
    dCSRmat * A,
    ivector * vertices,
    dCSRmat * P,
    AMG_param * param )
```

Energy-min interpolation.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix (index starts from 0)
<i>vertices</i>	Pointer to the indicator of CF splitting on fine or coarse grid
<i>P</i>	Pointer to the <a href="#">dCSRmat</a> matrix of resulted interpolation
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

#### Author

Shuo Zhang, Xuehai Huang

#### Date

04/04/2010

Modified by Chunsheng Feng, Zheng Li on 10/17/2012: add OMP support Modified by Chensong Zhang on 05/14/2013: reconstruct the code

Definition at line 63 of file PreAMGInterpEM.c.

## 9.67 PreAMGSetupCR.c File Reference

Brannick-Falgout compatible relaxation based AMG: SETUP phase.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [SHORT fasp\\_amg\\_setup\\_cr](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)

*Set up phase of Brannick Falgout CR coarsening for classic AMG.*

### 9.67.1 Detailed Description

Brannick-Falgout compatible relaxation based AMG: SETUP phase.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), and [PreAMGCoarsenCR.c](#)

Setup A, P, R and levels using the Compatible Relaxation coarsening for classic AMG interpolation

Reference: J. Brannick and R. Falgout Compatible relaxation and coarsening in AMG  
Copyright (C) 2010–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

TODO: Not working. Need to be fixed. –Chensong

## 9.67.2 Function Documentation

### 9.67.2.1 fasp\_amg\_setup\_cr()

```
SHORT fasp_amg_setup_cr (
    AMG_data * mgl,
    AMG_param * param )
```

Set up phase of Brannick Falgout CR coarsening for classic AMG.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

James Brannick

#### Date

04/21/2010

Modified by Chensong Zhang on 05/10/2013: adjust the structure.  
Definition at line 48 of file PreAMGSetupCR.c.

## 9.68 PreAMGSetupRS.c File Reference

Ruge-Stuben AMG: SETUP phase.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- [SHORT fasp\\_amg\\_setup\\_rs](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Setup phase of Ruge and Stuben's classic AMG.*

### 9.68.1 Detailed Description

Ruge-Stuben AMG: SETUP phase.

**Note**

This file contains Level-4 (Pre) functions. It requires: [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlalLUSetupCSR.c](#), [BlaSchwarzSetup.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), [PreAMGCoarsenRS.c](#), [PreAMGInterp.c](#), and [PreMGRecurAMLI.c](#)

Reference: Multigrid by U. Trottenberg, C. W. Oosterlee and A. Schuller Appendix P475 A.7 (by A. Brandt, P. Oswald and K. Stuben) Academic Press Inc., San Diego, CA, 2001.

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

## 9.68.2 Function Documentation

### 9.68.2.1 fasp\_amg\_setup\_rs()

```
SHORT fasp_amg_setup_rs (
    AMG_data * mgl,
    AMG_param * param )
```

Setup phase of Ruge and Stuben's classic AMG.

**Parameters**

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Chensong Zhang

**Date**

05/09/2010

Modified by Xiaozhe Hu on 01/23/2011: add AMLI cycle. Modified by Xiaozhe Hu on 04/24/2013: aggressive coarsening.

Modified by Chensong Zhang on 09/23/2014: check coarse spaces.

Definition at line 51 of file PreAMGSetupRS.c.

## 9.69 PreAMGSetupSA.c File Reference

Smoothed aggregation AMG: SETUP phase.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "PreAMGAggregation.inl"
#include "PreAMGAggregationCSR.inl"
```

## Functions

- [SHORT fasp\\_amg\\_setup\\_sa](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of smoothed aggregation AMG.*

### 9.69.1 Detailed Description

Smoothed aggregation AMG: SETUP phase.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxThreads.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlalLUSetupCSR.c](#), [BlaSchwarzSetup.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), and [PreMGRecurAMLI.c](#)

Setup A, P, PT and levels using the unsmoothed aggregation algorithm

Reference: P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994  
Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.69.2 Function Documentation

#### 9.69.2.1 fasp\_amg\_setup\_sa()

```
SHORT fasp_amg_setup_sa (
    AMG_data * mgl,
    AMG_param * param )
```

Set up phase of smoothed aggregation AMG.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Xiaozhe Hu

#### Date

09/29/2009

Modified by Xiaozhe Hu on 01/23/2011: add AMLI cycle. Modified by Chensong Zhang on 05/10/2013: adjust the structure.

Definition at line 63 of file PreAMGSetupSA.c.

## 9.70 PreAMGSetupSABSR.c File Reference

Smoothed aggregation AMG: SETUP phase (for BSR matrices)

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreAMGAggregation.inl"
#include "PreAMGAggregationBSR.inl"
#include "PreAMGAggregationUA.inl"
```

## Functions

- [SHORT fasp\\_amg\\_setup\\_sa\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)

*Set up phase of smoothed aggregation AMG (BSR format)*

### 9.70.1 Detailed Description

Smoothed aggregation AMG: SETUP phase (for BSR matrices)

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaFormat.c](#), [BlaLUSetupBSR.c](#), [BlaSmallMat.c](#), [BlaSparseBLC.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [BlaSpmvBSR.c](#), and [BlaSpmvCSR.c](#)

Setup A, P, PT and levels using the unsmoothed aggregation algorithm

Reference: P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994  
Copyright (C) 2014–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

### 9.70.2 Function Documentation

#### 9.70.2.1 fasp\_amg\_setup\_sa\_bsr()

```
INT fasp_amg_setup_sa_bsr (
    AMG\_data\_bsr * mgl,
    AMG\_param * param )
```

Set up phase of smoothed aggregation AMG (BSR format)

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data_bsr</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

#### Author

Xiaozhe Hu

Date

05/26/2014

Definition at line 61 of file PreAMGSetupSABSR.c.

## 9.71 PreAMGSetupUA.c File Reference

Unsmoothed aggregation AMG: SETUP phase.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreAMGAggregation.inl"
#include "PreAMGAggregationCSR.inl"
#include "PreAMGAggregationUA.inl"
```

### Functions

- [SHORT fasp\\_amg\\_setup\\_ua](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)

*Set up phase of unsmoothed aggregation AMG.*

#### 9.71.1 Detailed Description

Unsmoothed aggregation AMG: SETUP phase.

Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlallUSetupCSR.c](#), [BlaSchwarzSetup.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), and [PreMGRecurAMLI.c](#)

Setup A, P, PT and levels using the unsmoothed aggregation algorithm

Reference: A. Napov and Y. Notay An Algebraic Multigrid Method with Guaranteed Convergence Rate, 2012  
Copyright (C) 2011–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.71.2 Function Documentation

##### 9.71.2.1 fasp\_amg\_setup\_ua()

```
SHORT fasp_amg_setup_ua (
    AMG_data * mgl,
    AMG_param * param )
```

Set up phase of unsmoothed aggregation AMG.

Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

**Returns**

FASP\_SUCCESS if succeeded; otherwise, error information.

**Author**

Xiaozhe Hu

**Date**

12/28/2011

Definition at line 55 of file PreAMGSetupUA.c.

## 9.72 PreAMGSetupUABSR.c File Reference

Unsmoothed aggregation AMG: SETUP phase (for BSR matrices)

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreAMGAggregation.inl"
#include "PreAMGAggregationBSR.inl"
#include "PreAMGAggregationUA.inl"
```

**Functions**

- [SHORT fasp\\_amg\\_setup\\_ua\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of unsmoothed aggregation AMG (BSR format)*

### 9.72.1 Detailed Description

Unsmoothed aggregation AMG: SETUP phase (for BSR matrices)

**Note**

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaFormat.c](#), [BlalLUSetupBSR.c](#), [BlaSparseBLC.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), and [PreDataInit.c](#)

Setup A, P, PT and levels using the unsmoothed aggregation algorithm

Reference: P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994  
Copyright (C) 2012–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.72.2 Function Documentation

#### 9.72.2.1 fasp\_amg\_setup\_ua\_bsr()

```
INT fasp_amg_setup_ua_bsr (
    AMG_data_bsr * mgl,
    AMG_param * param )
```

Set up phase of unsmoothed aggregation AMG (BSR format)



## Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data_bsr</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

## Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

## Author

Xiaozhe Hu

## Date

03/16/2012

Definition at line 55 of file PreAMGSetupUABSR.c.

## 9.73 PreBLC.c File Reference

Preconditioners for [dBLMat](#) matrices.

```
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_precond\\_dbc\\_diag\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*Block diagonal preconditioner (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_diag\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*Block diagonal preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_diag\\_4](#) (REAL \*r, REAL \*z, void \*data)  
*Block diagonal preconditioning (4x4 blocks)*
- void [fasp\\_precond\\_dbc\\_lower\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_lower\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_lower\\_4](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (4x4 blocks)*
- void [fasp\\_precond\\_dbc\\_upper\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block upper triangular preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_upper\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block upper triangular preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_SGS\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*Block symmetric GS preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_SGS\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*Block symmetric GS preconditioning (3x3 blocks)*
- void [fasp\\_precond\\_dbc\\_sweeping](#) (REAL \*r, REAL \*z, void \*data)  
*Sweeping preconditioner for Maxwell equations.*

### 9.73.1 Detailed Description

Preconditioners for [dBLCmat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxVector.c](#), [BlaSpmvCSR.c](#), and [PreMGCycle.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

TODO: Separate solve and setup phases for direct solvers!!! –Chensong

### 9.73.2 Function Documentation

#### 9.73.2.1 `fasp_precond_dblc_diag_3()`

```
void fasp_precond_dblc_diag_3 (
    REAL * r,
    REAL * z,
    void * data )
```

Block diagonal preconditioner (3x3 blocks)

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Xiaozhe Hu

#### Date

07/10/2014

#### Note

Each diagonal block is solved exactly

Definition at line 38 of file PreBLC.c.

#### 9.73.2.2 `fasp_precond_dblc_diag_3_amg()`

```
void fasp_precond_dblc_diag_3_amg (
    REAL * r,
    REAL * z,
    void * data )
```

Block diagonal preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

Each diagonal block is solved by AMG

Definition at line 126 of file PreBLC.c.

**9.73.2.3 fasp\_precond\_dbldc\_diag\_4()**

```
void fasp_precond_dbldc_diag_4 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Block diagonal preconditioning (4x4 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

Each diagonal block is solved exactly

Definition at line 191 of file PreBLC.c.

**9.73.2.4 fasp\_precond\_dbldc\_lower\_3()**

```
void fasp_precond_dbldc_lower_3 (  
    REAL * r,
```

```
    REAL * z,  
    void * data )
```

block lower triangular preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

Each diagonal block is solved exactly

Definition at line 291 of file PreBLC.c.

**9.73.2.5 fasp\_precond\_dbldc\_lower\_3\_amg()**

```
void fasp_precond_dbldc_lower_3_amg (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block lower triangular preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

Each diagonal block is solved by AMG

Definition at line 379 of file PreBLC.c.

**9.73.2.6 fasp\_precond\_dbldc\_lower\_4()**

```
void fasp_precond_dbldc_lower_4 (  
    REAL * r,
```

```
    REAL * z,  
    void * data )
```

block lower triangular preconditioning (4x4 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

Each diagonal block is solved exactly

Definition at line 453 of file PreBLC.c.

**9.73.2.7 fasp\_precond\_dblc\_SGS\_3()**

```
void fasp_precond_dblc_SGS_3 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Block symmetric GS preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

02/19/2015

## Note

Each diagonal block is solved exactly

Definition at line 725 of file PreBLC.c.

**9.73.2.8 fasp\_precond\_dblc\_SGS\_3\_amg()**

```
void fasp_precond_dblc_SGS_3_amg (  
    REAL * r,
```

```
    REAL * z,  
    void * data )
```

Block symmetric GS preconditioning (3x3 blocks)



**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu

**Date**

02/19/2015

**Note**

Each diagonal block is solved by AMG

Definition at line 838 of file PreBLC.c.

**9.73.2.9 fasp\_precond\_dblc\_sweeping()**

```
void fasp_precond_dblc_sweeping (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Sweeping preconditioner for Maxwell equations.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu

**Date**

05/01/2014

**Note**

Each diagonal block is solved exactly

Definition at line 939 of file PreBLC.c.

**9.73.2.10 fasp\_precond\_dblc\_upper\_3()**

```
void fasp_precond_dblc_upper_3 (  
    REAL * r,
```

```
    REAL * z,  
    void * data )
```

block upper triangular preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

02/18/2015

## Note

Each diagonal block is solved exactly

Definition at line 557 of file PreBLC.c.

**9.73.2.11 fasp\_precond\_dblc\_upper\_3\_amg()**

```
void fasp_precond_dblc_upper_3_amg (
    REAL * r,
    REAL * z,
    void * data )
```

block upper triangular preconditioning (3x3 blocks)

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

02/19/2015

## Note

Each diagonal block is solved by AMG

Definition at line 645 of file PreBLC.c.

**9.74 PreBSR.c File Reference**

Preconditioners for dBSRmat matrices.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "PreMGUtil.inl"
```

## Functions

- void [fasp\\_precond\\_dbsr\\_diag](#) (REAL \*r, REAL \*z, void \*data)  
*Diagonal preconditioner  $z=inv(D)*r$ .*
- void [fasp\\_precond\\_dbsr\\_diag\\_nc2](#) (REAL \*r, REAL \*z, void \*data)  
*Diagonal preconditioner  $z=inv(D)*r$ .*
- void [fasp\\_precond\\_dbsr\\_diag\\_nc3](#) (REAL \*r, REAL \*z, void \*data)  
*Diagonal preconditioner  $z=inv(D)*r$ .*
- void [fasp\\_precond\\_dbsr\\_diag\\_nc5](#) (REAL \*r, REAL \*z, void \*data)  
*Diagonal preconditioner  $z=inv(D)*r$ .*
- void [fasp\\_precond\\_dbsr\\_diag\\_nc7](#) (REAL \*r, REAL \*z, void \*data)  
*Diagonal preconditioner  $z=inv(D)*r$ .*
- void [fasp\\_precond\\_dbsr\\_ilu](#) (REAL \*r, REAL \*z, void \*data)  
*ILU preconditioner.*
- void [fasp\\_precond\\_dbsr\\_ilu\\_mc\\_omp](#) (REAL \*r, REAL \*z, void \*data)  
*Multi-thread Parallel ILU preconditioner based on graph coloring.*
- void [fasp\\_precond\\_dbsr\\_ilu\\_ls\\_omp](#) (REAL \*r, REAL \*z, void \*data)  
*Multi-thread Parallel ILU preconditioner based on level schedule strategy.*
- void [fasp\\_precond\\_dbsr\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*AMG preconditioner.*
- void [fasp\\_precond\\_dbsr\\_amg\\_nk](#) (REAL \*r, REAL \*z, void \*data)  
*AMG with extra near kernel solve preconditioner.*
- void [fasp\\_precond\\_dbsr\\_namli](#) (REAL \*r, REAL \*z, void \*data)  
*Nonlinear AMLI-cycle AMG preconditioner.*

### 9.74.1 Detailed Description

Preconditioners for [dBSRmat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxParam.c](#), [AuxThreads.c](#), [AuxVector.c](#), [BlaSmallMat.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [KrySPcg.c](#), [KrySPvmres.c](#), [PreMGCycle.c](#), and [PreMGRecurAMLI.c](#)

---

Copyright (C) 2010–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.74.2 Function Documentation

#### 9.74.2.1 [fasp\\_precond\\_dbsr\\_amg\(\)](#)

```
void fasp_precond_dbsr_amg (
    REAL * r,
    REAL * z,
    void * data )
```

AMG preconditioner.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu

**Date**

08/07/2011

Definition at line 986 of file PreBSR.c.

**9.74.2.2 fasp\_precond\_dbsr\_amg\_nk()**

```
void fasp_precond_dbsr_amg_nk (  
    REAL * r,  
    REAL * z,  
    void * data )
```

AMG with extra near kernel solve preconditioner.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu

**Date**

05/26/2014

Definition at line 1030 of file PreBSR.c.

**9.74.2.3 fasp\_precond\_dbsr\_diag()**

```
void fasp_precond_dbsr_diag (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Zhou Zhiyang, Xiaozhe Hu

**Date**

10/26/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/24/2012

**Note**

Works for general nb (Xiaozhe)

Definition at line 49 of file PreBSR.c.

**9.74.2.4 fasp\_precond\_dbsr\_diag\_nc2()**

```
void fasp_precond_dbsr_diag_nc2 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Zhou Zhiyang, Xiaozhe Hu

**Date**

11/18/2011

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/24/2012

**Note**

Works for 2-component (Xiaozhe)

Definition at line 121 of file PreBSR.c.

**9.74.2.5 fasp\_precond\_dbsr\_diag\_nc3()**

```
void fasp_precond_dbsr_diag_nc3 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Zhou Zhiyang, Xiaozhe Hu

**Date**

01/06/2011

Modified by Chunsheng Feng Xiaoqiang Yue on 05/24/2012

**Note**

Works for 3-component (Xiaozhe)

Definition at line 169 of file PreBSR.c.

**9.74.2.6 fasp\_precond\_dbsr\_diag\_nc5()**

```
void fasp_precond_dbsr_diag_nc5 (
    REAL * r,
    REAL * z,
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Zhou Zhiyang, Xiaozhe Hu

**Date**

01/06/2011

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/24/2012

**Note**

Works for 5-component (Xiaozhe)

Definition at line 217 of file PreBSR.c.

**9.74.2.7 fasp\_precond\_dbsr\_diag\_nc7()**

```
void fasp_precond_dbsr_diag_nc7 (
    REAL * r,
    REAL * z,
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Zhou Zhiyang, Xiaozhe Hu

**Date**

01/06/2011

Modified by Chunsheng Feng Xiaoqiang Yue on 05/24/2012

**Note**

Works for 7-component (Xiaozhe)

Definition at line 265 of file PreBSR.c.

**9.74.2.8 fasp\_precond\_dbsr\_ilu()**

```
void fasp_precond_dbsr_ilu (
    REAL * r,
    REAL * z,
    void * data )
```

ILU preconditioner.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

11/09/2010

**Note**

Works for general nb (Xiaozhe)

Definition at line 311 of file PreBSR.c.

**9.74.2.9 fasp\_precond\_dbsr\_ilu\_ls\_omp()**

```
void fasp_precond_dbsr_ilu_ls_omp (
    REAL * r,
```



```
    REAL * z,  
    void * data )
```

Multi-thread Parallel ILU preconditioner based on level schedule strategy.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Zheng Li

#### Date

12/04/2016

#### Note

Only works for nb 1, 2, and 3 (Zheng)

Definition at line 773 of file PreBSR.c.

#### 9.74.2.10 fasp\_precond\_dbsr\_ilu\_mc\_omp()

```
void fasp_precond_dbsr_ilu_mc_omp (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Multi-thread Parallel ILU preconditioner based on graph coloring.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Zheng Li

#### Date

12/04/2016

#### Note

Only works for nb 1, 2, and 3 (Zheng)

Definition at line 569 of file PreBSR.c.

### 9.74.2.11 fasp\_precond\_dbsr\_namli()

```
void fasp_precond_dbsr_namli (
    REAL * r,
    REAL * z,
    void * data )
```

Nonlinear AMLI-cycle AMG preconditioner.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Xiaozhe Hu

#### Date

02/06/2012

Definition at line 1124 of file PreBSR.c.

## 9.75 PreCSR.c File Reference

Preconditioners for dCSRmat matrices.

```
#include "fasp.h"
#include "fasp_funcs.h"
#include "PreMGUtil.inl"
```

### Functions

- `precond * fasp_precond_setup` (const `SHORT` `precond_type`, `AMG_param` \*`amgparam`, `ILU_param` \*`iluparam`, `dCSRmat` \*`A`)  
*Setup preconditioner interface for iterative methods.*
- void `fasp_precond_diag` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*Diagonal preconditioner  $z = \text{inv}(D) * r$ .*
- void `fasp_precond_ilu` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*ILU preconditioner.*
- void `fasp_precond_ilu_forward` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*ILU preconditioner: only forward sweep.*
- void `fasp_precond_ilu_backward` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*ILU preconditioner: only backward sweep.*
- void `fasp_precond_swz` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*get z from r by Schwarz*
- void `fasp_precond_amg` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*AMG preconditioner.*
- void `fasp_precond_famg` (`REAL` \*`r`, `REAL` \*`z`, void \*`data`)  
*Full AMG preconditioner.*

- void [fasp\\_precond\\_amli](#) (REAL \*r, REAL \*z, void \*data)  
*AMLI AMG preconditioner.*
- void [fasp\\_precond\\_namli](#) (REAL \*r, REAL \*z, void \*data)  
*Nonlinear AMLI AMG preconditioner.*
- void [fasp\\_precond\\_amg\\_nk](#) (REAL \*r, REAL \*z, void \*data)  
*AMG with extra near kernel solve as preconditioner.*

### 9.75.1 Detailed Description

Preconditioners for [dCSRmat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxParam.c](#), [AuxVector.c](#), [BlalLUSetupCSR.c](#), [BlaSchwarzSetup.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), [KrySPcg.c](#), [KrySPvgmres.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreDataInit.c](#), [PreMGCycle.c](#), [PreMGCycleFull.c](#), and [PreMGRecurAMLI.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.75.2 Function Documentation

#### 9.75.2.1 [fasp\\_precond\\_amg\(\)](#)

```
void fasp_precond_amg (
    REAL * r,
    REAL * z,
    void * data )
```

AMG preconditioner.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Chensong Zhang

#### Date

04/06/2010

Definition at line 416 of file PreCSR.c.

#### 9.75.2.2 [fasp\\_precond\\_amg\\_nk\(\)](#)

```
void fasp_precond_amg_nk (
    REAL * r,
```

```

    REAL * z,
    void * data )

```

AMG with extra near kernel solve as preconditioner.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Xiaozhe Hu

#### Date

05/26/2014

Definition at line 548 of file PreCSR.c.

### 9.75.2.3 fasp\_precond\_amli()

```

void fasp_precond_amli (
    REAL * r,
    REAL * z,
    void * data )

```

AMLI AMG preconditioner.

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Xiaozhe Hu

#### Date

01/23/2011

Definition at line 482 of file PreCSR.c.

### 9.75.2.4 fasp\_precond\_diag()

```

void fasp_precond_diag (
    REAL * r,
    REAL * z,
    void * data )

```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Chensong Zhang

## Date

04/06/2010

Definition at line 172 of file PreCSR.c.

### 9.75.2.5 fasp\_precond\_famg()

```
void fasp_precond_famg (  
    REAL * r,  
    REAL * z,  
    void * data )
```

Full AMG preconditioner.

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

02/27/2011

Definition at line 449 of file PreCSR.c.

### 9.75.2.6 fasp\_precond\_ilu()

```
void fasp_precond_ilu (  
    REAL * r,  
    REAL * z,  
    void * data )
```

ILU preconditioner.

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Shiquan Zhang

**Date**

04/06/2010

Definition at line 198 of file PreCSR.c.

**9.75.2.7 fasp\_precond\_ilu\_backward()**

```
void fasp_precond_ilu_backward (
    REAL * r,
    REAL * z,
    void * data )
```

ILU preconditioner: only backward sweep.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

04/06/2010

Definition at line 317 of file PreCSR.c.

**9.75.2.8 fasp\_precond\_ilu\_forward()**

```
void fasp_precond_ilu_forward (
    REAL * r,
    REAL * z,
    void * data )
```

ILU preconditioner: only forward sweep.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu, Shiquan Zhang

## Date

04/06/2010

Definition at line 263 of file PreCSR.c.

**9.75.2.9 fasp\_precond\_namli()**

```
void fasp_precond_namli (
    REAL * r,
    REAL * z,
    void * data )
```

Nonlinear AMLI AMG preconditioner.

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Xiaozhe Hu

## Date

04/25/2011

Definition at line 515 of file PreCSR.c.

**9.75.2.10 fasp\_precond\_setup()**

```
precond * fasp_precond_setup (
    const SHORT precondition_type,
    AMG_param * amgparam,
    ILU_param * iluparam,
    dCSRmat * A )
```

Setup preconditioner interface for iterative methods.

## Parameters

<i>precond_type</i>	Preconditioner type
<i>amgparam</i>	Pointer to AMG parameters
<i>iluparam</i>	Pointer to ILU parameters
<i>A</i>	Pointer to the coefficient matrix

## Returns

Pointer to preconditioner

**Author**

Feiteng Huang

**Date**

05/18/2009

Definition at line 46 of file PreCSR.c.

**9.75.2.11 fasp\_precond\_swz()**

```
void fasp_precond_swz (
    REAL * r,
    REAL * z,
    void * data )
```

get z from r by Schwarz

**Parameters**

<i>r</i>	Pointer to residual
<i>z</i>	Pointer to preconditioned residual
<i>data</i>	Pointer to precondition data

**Author**

Xiaozhe Hu

**Date**

03/22/2010

**Note**

Change Schwarz interface by Zheng Li on 11/18/2014

Definition at line 371 of file PreCSR.c.

**9.76 PreDataInit.c File Reference**

Initialize important data structures.

```
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- void [fasp\\_precond\\_data\\_init](#) ([precond\\_data](#) \*pcdata)  
*Initialize [precond\\_data](#).*
- [AMG\\_data](#) \* [fasp\\_amg\\_data\\_create](#) ([SHORT](#) max\_levels)  
*Create and initialize [AMG\\_data](#) for classical and SA AMG.*
- void [fasp\\_amg\\_data\\_free](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Free [AMG\\_data](#) data memeory space.*



- [AMG\\_data\\_bsr](#) \* [fasp\\_amg\\_data\\_bsr\\_create](#) ([SHORT](#) max\_levels)  
*Create and initialize [AMG\\_data](#) data sturcture for AMG/SAMG (BSR format)*
- void [fasp\\_amg\\_data\\_bsr\\_free](#) ([AMG\\_data\\_bsr](#) \*mgl)  
*Free [AMG\\_data\\_bsr](#) data memeory space.*
- void [fasp\\_ilu\\_data\\_create](#) (const [INT](#) iwk, const [INT](#) nwork, [ILU\\_data](#) \*iludata)  
*Allocate workspace for ILU factorization.*
- void [fasp\\_ilu\\_data\\_free](#) ([ILU\\_data](#) \*iludata)  
*Create [ILU\\_data](#) sturcture.*
- void [fasp\\_swz\\_data\\_free](#) ([SWZ\\_data](#) \*swzdata)  
*Free [SWZ\\_data](#) data memeory space.*

### 9.76.1 Detailed Description

Initialize important data structures.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxMemory.c](#), [AuxVector.c](#), [BlaSparseBSR.c](#), and [BlaSparseCSR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

**Released under the terms of the GNU Lesser General Public License 3.0 or later.**

#### Warning

Every structures should be initialized before usage.

### 9.76.2 Function Documentation

#### 9.76.2.1 [fasp\\_amg\\_data\\_bsr\\_create\(\)](#)

```
AMG\_data\_bsr * fasp\_amg\_data\_bsr\_create (
    SHORT max_levels )
```

Create and initialize [AMG\\_data](#) data sturcture for AMG/SAMG (BSR format)

#### Parameters

<i>max_levels</i>	Max number of levels allowed
-------------------	------------------------------

#### Returns

Pointer to the [AMG\\_data](#) data structure

#### Author

Xiaozhe Hu

#### Date

08/07/2011

Definition at line 178 of file PreDataInit.c.

### 9.76.2.2 fasp\_amg\_data\_bsr\_free()

```
void fasp_amg_data_bsr_free (
    AMG_data_bsr * mgl )
```

Free [AMG\\_data\\_bsr](#) data memeory space.

#### Parameters

<i>mgl</i>	Pointer to the <a href="#">AMG_data_bsr</a>
------------	---

#### Author

Xiaozhe Hu, Chensong Zhang

#### Date

2013/02/13

Modified by Chensong Zhang on 08/14/2017: Check for max\_levels == 1  
Definition at line 210 of file PreDataInit.c.

### 9.76.2.3 fasp\_amg\_data\_create()

```
AMG_data * fasp_amg_data_create (
    SHORT max_levels )
```

Create and initialize [AMG\\_data](#) for classical and SA AMG.

#### Parameters

<i>max_levels</i>	Max number of levels allowed
-------------------	------------------------------

#### Returns

Pointer to the [AMG\\_data](#) data structure

#### Author

Chensong Zhang

#### Date

2010/04/06

Definition at line 64 of file PreDataInit.c.

### 9.76.2.4 fasp\_amg\_data\_free()

```
void fasp_amg_data_free (
    AMG_data * mgl,
    AMG_param * param )
```

Free [AMG\\_data](#) data memeory space.

## Parameters

<i>mgl</i>	Pointer to the <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters

## Author

Chensong Zhang

## Date

2010/04/06

Modified by Chensong Zhang on 05/05/2013: Clean up param as well! Modified by Hongxuan Zhang on 12/15/2015: Free memory for Intel MKL PARDISO Modified by Chunsheng Feng on 02/12/2017: Permute A back to its origin for ILUtp Modified by Chunsheng Feng on 08/11/2017: Check for max\_levels == 1  
Definition at line 98 of file PreDataInit.c.

### 9.76.2.5 fasp\_ilu\_data\_create()

```
void fasp_ilu_data_create (
    const INT iwk,
    const INT nwork,
    ILU_data * iludata )
```

Allocate workspace for ILU factorization.

## Parameters

<i>iwk</i>	Size of the index array
<i>nwork</i>	Size of the work array
<i>iludata</i>	Pointer to the <a href="#">ILU_data</a>

## Author

Chensong Zhang

## Date

2010/04/06

Modified by Chunsheng Feng on 02/12/2017: add iperm array for ILUtp  
Definition at line 262 of file PreDataInit.c.

### 9.76.2.6 fasp\_ilu\_data\_free()

```
void fasp_ilu_data_free (
    ILU_data * iludata )
```

Create [ILU\\_data](#) sturcture.

## Parameters

<i>iludata</i>	Pointer to <a href="#">ILU_data</a>
----------------	-------------------------------------

**Author**

Chensong Zhang

**Date**

2010/04/03

Modified by Chunsheng Feng on 02/12/2017: add iperm array for ILUtp  
Definition at line 297 of file PreDataInit.c.

**9.76.2.7 fasp\_precond\_data\_init()**

```
void fasp_precond_data_init (
    precondition_data * pcd_data )
```

Initialize `precond_data`.

**Parameters**

<code>pcdata</code>	Preconditioning data structure
---------------------	--------------------------------

**Author**

Chensong Zhang

**Date**

2010/03/23

Definition at line 33 of file PreDataInit.c.

**9.76.2.8 fasp\_swz\_data\_free()**

```
void fasp_swz_data_free (
    SWZ_data * swzdata )
```

Free `SWZ_data` data memory space.

**Parameters**

<code>swzdata</code>	Pointer to the <code>SWZ_data</code> for Schwarz methods
----------------------	--

**Author**

Xiaozhe Hu

**Date**

2010/04/06

Definition at line 338 of file PreDataInit.c.

**9.77 PreMGCycle.c File Reference**

Abstract multigrid cycle – non-recursive version.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreMGUtil.inl"
#include "PreMGSmother.inl"
```

## Functions

- void [fasp\\_solver\\_mgcycle](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Solve  $Ax=b$  with non-recursive multigrid cycle.*
- void [fasp\\_solver\\_mgcycle\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Solve  $Ax=b$  with non-recursive multigrid cycle.*

### 9.77.1 Detailed Description

Abstract multigrid cycle – non-recursive version.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaArray.c](#), [BlaSchwarzSetup.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [ltrSmootherBSR.c](#), [ltrSmootherCSR.c](#), [ltrSmootherCSRpoly.c](#), [KryPcg.c](#), [KryPvgmres.c](#), [KrySPcg.c](#), and [KrySPvgmres.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.77.2 Function Documentation

#### 9.77.2.1 [fasp\\_solver\\_mgcycle\(\)](#)

```
void fasp_solver_mgcycle (
    AMG\_data * mgl,
    AMG\_param * param )
```

Solve  $Ax=b$  with non-recursive multigrid cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Author

Chensong Zhang

#### Date

10/06/2010

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Chensong Zhang on 12/30/2014: update Schwarz smoothers.

Definition at line 48 of file PreMGCycle.c.

### 9.77.2.2 fasp\_solver\_mgcycle\_bsr()

```
void fasp_solver_mgcycle_bsr (
    AMG_data_bsr * mgl,
    AMG_param * param )
```

Solve  $Ax=b$  with non-recursive multigrid cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data_bsr</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Author

Xiaozhe Hu

#### Date

08/07/2011

Definition at line 259 of file PreMGCycle.c.

## 9.78 PreMGCycleFull.c File Reference

Abstract non-recursive full multigrid cycle.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreMGUtil.inl"
#include "PreMGSmother.inl"
```

### Functions

- void [fasp\\_solver\\_fmecycle](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Solve  $Ax=b$  with non-recursive full multigrid K-cycle.*

### 9.78.1 Detailed Description

Abstract non-recursive full multigrid cycle.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaSchwarzSetup.c](#), [BlaArray.c](#), [BlaSpmvCSR.c](#), [BlaVector.c](#), [ItrSmootherCSR.c](#), [ItrSmootherCSRpoly.c](#), [KryPcg.c](#), [KrySPcg.c](#), and [KrySPvgmres.c](#)

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.78.2 Function Documentation

### 9.78.2.1 fasp\_solver\_fmecycle()

```
void fasp_solver_fmecycle (
    AMG_data * mgl,
    AMG_param * param )
```

Solve  $Ax=b$  with non-recursive full multigrid K-cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Author

Chensong Zhang

#### Date

02/27/2011

Modified by Chensong Zhang on 06/01/2012: fix a bug when there is only one level. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 47 of file PreMGCycleFull.c.

## 9.79 PreMGRecur.c File Reference

Abstract multigrid cycle – recursive version.

```
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "PreMGUtil.inl"
#include "PreMGSmother.inl"
```

### Functions

- void [fasp\\_solver\\_mgrecur](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param, INT level)  
*Solve  $Ax=b$  with recursive multigrid K-cycle.*

### 9.79.1 Detailed Description

Abstract multigrid cycle – recursive version.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaSpmvCSR.c](#), [ItrSmootherCSR.c](#), [ItrSmootherCSRpoly.c](#), [KryPcg.c](#), [KrySPcg.c](#), and [KrySPvgmres.c](#)

#### Warning

Not used any more! Deprecated in the future versions.

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.79.2 Function Documentation

### 9.79.2.1 fasp\_solver\_mgrecur()

```
void fasp_solver_mgrecur (
    AMG_data * mgl,
    AMG_param * param,
    INT level )
```

Solve  $Ax=b$  with recursive multigrid K-cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>
<i>level</i>	Index of the current level

#### Author

Xuehai Huang, Chensong Zhang

#### Date

04/06/2010

Modified by Chensong Zhang on 02/27/2013: update direct solvers.  
Definition at line 47 of file PreMGRecur.c.

## 9.80 PreMGRecurAMLI.c File Reference

Abstract AMLI multilevel iteration – recursive version.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreMGUtil.inl"
#include "PreMGSmoothing.inl"
#include "PreMGRecurAMLI.inl"
```

### Functions

- void [fasp\\_solver\\_amli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) l)  
*Solve  $Ax=b$  with recursive AMLI-cycle.*
- void [fasp\\_solver\\_namli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) l, [INT](#) num\_levels)  
*Solve  $Ax=b$  with recursive nonlinear AMLI-cycle.*
- void [fasp\\_solver\\_namli\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) l, [INT](#) num\_levels)  
*Solve  $Ax=b$  with recursive nonlinear AMLI-cycle.*
- void [fasp\\_amg\\_amli\\_coef](#) (const [REAL](#) lambda\_max, const [REAL](#) lambda\_min, const [INT](#) degree, [REAL](#) \*coef)  
*Compute the coefficients of the polynomial used by AMLI-cycle.*



### 9.80.1 Detailed Description

Abstract AMLI multilevel iteration – recursive version.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxParam.c](#), [AuxVector.c](#), [BlaSchwarzSetup.c](#), [BlaArray.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [ItrSmootherBSR.c](#), [ItrSmootherCSR.c](#), [ItrSmootherCSRpoly.c](#), [KryPcg.c](#), [KryPvfgmres.c](#), [KrySPcg.c](#), [KrySPvfgmres.c](#), [PreBSR.c](#), and [PreCSR.c](#)

This file includes both AMLI and non-linear AMLI cycles

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.80.2 Function Documentation

#### 9.80.2.1 fasp\_amg\_amli\_coef()

```
void fasp_amg_amli_coef (
    const REAL lambda_max,
    const REAL lambda_min,
    const INT degree,
    REAL * coef )
```

Compute the coefficients of the polynomial used by AMLI-cycle.

#### Parameters

<i>lambda_max</i>	Maximal lambda
<i>lambda_min</i>	Minimal lambda
<i>degree</i>	Degree of polynomial approximation
<i>coef</i>	Coefficient of AMLI (output)

#### Author

Xiaozhe Hu

#### Date

01/23/2011

Definition at line 699 of file PreMGRecurAMLI.c.

#### 9.80.2.2 fasp\_solver\_amli()

```
void fasp_solver_amli (
    AMG_data * mgl,
    AMG_param * param,
    INT l )
```

Solve  $Ax=b$  with recursive AMLI-cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
------------	---

**Parameters**

<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>
<i>l</i>	Current level

**Author**

Xiaozhe Hu

**Date**

01/23/2011

**Note**

AMLI polynomial computed by the best approximation of  $1/x$ . Refer to Johannes K. Kraus, Panayot S. Vassilevski, Ludmil T. Zikatanov, "Polynomial of best uniform approximation to  $x^{-1}$  and smoothing in two-level methods", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Zheng Li on 11/10/2014: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.  
Definition at line 58 of file PreMGRecurAMLI.c.

**9.80.2.3 fasp\_solver\_namli()**

```
void fasp_solver_namli (
    AMG_data * mgl,
    AMG_param * param,
    INT l,
    INT num_levels )
```

Solve  $Ax=b$  with recursive nonlinear AMLI-cycle.

**Parameters**

<i>mgl</i>	Pointer to <a href="#">AMG_data</a> data
<i>param</i>	Pointer to AMG parameters
<i>l</i>	Current level
<i>num_levels</i>	Total number of levels

**Author**

Xiaozhe Hu

**Date**

04/06/2010

**Note**

Refer to Xiazhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Zheng Li on 11/10/2014: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.  
Definition at line 275 of file PreMGRecurAMLI.c.

### 9.80.2.4 fasp\_solver\_namli\_bsr()

```
void fasp_solver_namli_bsr (
    AMG_data_bsr * mgl,
    AMG_param * param,
    INT l,
    INT num_levels )
```

Solve  $Ax=b$  with recursive nonlinear AMLI-cycle.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>
<i>l</i>	Current level
<i>num_levels</i>	Total number of levels

#### Author

Xiaozhe Hu

#### Date

04/06/2010

#### Note

Nonlinear AMLI-cycle. Refer to Xiazhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 501 of file PreMGRecurAMLI.c.

## 9.81 PreMGSolve.c File Reference

Algebraic multigrid iterations: SOLVE phase.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_amg\\_solve](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*AMG – SOLVE phase.*
- [INT fasp\\_amg\\_solve\\_amli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*AMLI – SOLVE phase.*
- [INT fasp\\_amg\\_solve\\_namli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Nonlinear AMLI – SOLVE phase.*
- void [fasp\\_famg\\_solve](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*FMG – SOLVE phase.*

### 9.81.1 Detailed Description

Algebraic multigrid iterations: SOLVE phase.

#### Note

Solve  $Ax=b$  using multigrid method. This is SOLVE phase only and is independent of SETUP method used! Should be called after multigrid hierarchy has been generated!

This file contains Level-4 (Pre) functions. It requires: [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSpmvCSR.c](#), [BlaVector.c](#), [PreMGCycle.c](#), [PreMGCycleFull.c](#), and [PreMGRecurAMLI.c](#)

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.81.2 Function Documentation

#### 9.81.2.1 fasp\_amg\_solve()

```
INT fasp_amg_solve (
    AMG_data * mgl,
    AMG_param * param )
```

AMG – SOLVE phase.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xuehai Huang, Chensong Zhang

#### Date

04/02/2010

Modified by Chensong 04/21/2013: Fix an output typo  
Definition at line 49 of file PreMGSolve.c.

#### 9.81.2.2 fasp\_amg\_solve\_amli()

```
INT fasp_amg_solve_amli (
    AMG_data * mgl,
    AMG_param * param )
```

AMLI – SOLVE phase.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

01/23/2011

Modified by Chensong 04/21/2013: Fix an output typo

**Note**

AMLI polynomial computed by the best approximation of  $1/x$ . Refer to Johannes K. Kraus, Panayot S. Vassilevski, Ludmil T. Zikatanov, "Polynomial of best uniform approximation to  $x^{-1}$  and smoothing in two-level methods", 2013.

Definition at line 142 of file PreMGSolve.c.

**9.81.2.3 fasp\_amg\_solve\_namli()**

```
INT fasp_amg_solve_namli (
    AMG_data * mgl,
    AMG_param * param )
```

Nonlinear AMLI – SOLVE phase.

**Parameters**

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

04/30/2011

Modified by Chensong 04/21/2013: Fix an output typo

**Note**

Nonlinear AMLI-cycle.

Refer to Xiaozhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid", 2013.

Definition at line 230 of file PreMGSolve.c.

### 9.81.2.4 fasp\_famg\_solve()

```
void fasp_famg_solve (
    AMG_data * mgl,
    AMG_param * param )
```

FMG – SOLVE phase.

#### Parameters

<i>mgl</i>	Pointer to AMG data: <a href="#">AMG_data</a>
<i>param</i>	Pointer to AMG parameters: <a href="#">AMG_param</a>

#### Author

Chensong Zhang

#### Date

01/10/2012

Definition at line 308 of file PreMGsSolve.c.

## 9.82 PreSTR.c File Reference

Preconditioners for [dSTRmat](#) matrices.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_precond\\_dstr\\_diag](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Diagonal preconditioner  $z = \text{inv}(D) * r$ .*
- void [fasp\\_precond\\_dstr\\_ilu0](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition.*
- void [fasp\\_precond\\_dstr\\_ilu1](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition.*
- void [fasp\\_precond\\_dstr\\_ilu0\\_forward](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition:  $Lz = r$ .*
- void [fasp\\_precond\\_dstr\\_ilu0\\_backward](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition:  $Uz = r$ .*
- void [fasp\\_precond\\_dstr\\_ilu1\\_forward](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition:  $Lz = r$ .*
- void [fasp\\_precond\\_dstr\\_ilu1\\_backward](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition:  $Uz = r$ .*
- void [fasp\\_precond\\_dstr\\_blockgs](#) ([REAL](#) \*r, [REAL](#) \*z, void \*data)  
*CPR-type preconditioner (STR format)*

### 9.82.1 Detailed Description

Preconditioners for [dSTRmat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxVector.c](#), [BlaSmallMat.c](#), [BlaArray.c](#), and [ltrSmootherSTR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.82.2 Function Documentation

#### 9.82.2.1 fasp\_precond\_dstr\_blockgs()

```
void fasp_precond_dstr_blockgs (
    REAL * r,
    REAL * z,
    void * data )
```

CPR-type preconditioner (STR format)

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Shiquan Zhang

#### Date

10/17/2010

Definition at line 1715 of file PreSTR.c.

#### 9.82.2.2 fasp\_precond\_dstr\_diag()

```
void fasp_precond_dstr_diag (
    REAL * r,
    REAL * z,
    void * data )
```

Diagonal preconditioner  $z = \text{inv}(D) * r$ .

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Shiquan Zhang

**Date**

04/06/2010

Definition at line 44 of file PreSTR.c.

**9.82.2.3 fasp\_precond\_dstr\_ilu0()**

```
void fasp_precond_dstr_ilu0 (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(0) decomposition.

**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Shiquan Zhang

**Date**

04/21/2010

Definition at line 71 of file PreSTR.c.

**9.82.2.4 fasp\_precond\_dstr\_ilu0\_backward()**

```
void fasp_precond_dstr_ilu0_backward (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(0) decomposition:  $Uz = r$ .**Parameters**

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

**Author**

Shiquan Zhang



## Date

06/07/2010

Definition at line 987 of file PreSTR.c.

**9.82.2.5 fasp\_precond\_dstr\_ilu0\_forward()**

```
void fasp_precond_dstr_ilu0_forward (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(0) decomposition:  $Lz = r$ .

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Shiquan Zhang

## Date

06/07/2010

Definition at line 824 of file PreSTR.c.

**9.82.2.6 fasp\_precond\_dstr\_ilu1()**

```
void fasp_precond_dstr_ilu1 (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(1) decomposition.

## Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

## Author

Shiquan Zhang

## Date

04/21/2010

Definition at line 349 of file PreSTR.c.

### 9.82.2.7 fasp\_precond\_dstr\_ilu1\_backward()

```
void fasp_precond_dstr_ilu1_backward (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(1) decomposition:  $Uz = r$ .

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Shiquan Zhang

#### Date

04/21/2010

Definition at line 1434 of file PreSTR.c.

### 9.82.2.8 fasp\_precond\_dstr\_ilu1\_forward()

```
void fasp_precond_dstr_ilu1_forward (
    REAL * r,
    REAL * z,
    void * data )
```

Preconditioning using STR\_ILU(1) decomposition:  $Lz = r$ .

#### Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

#### Author

Shiquan Zhang

#### Date

04/21/2010

Definition at line 1168 of file PreSTR.c.

## 9.83 SolAMG.c File Reference

AMG method as an iterative solver.

```
#include <time.h>
#include "fasp.h"
```

```
#include "fasp_funcs.h"
```

## Functions

- [INT fasp\\_solver\\_amg](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [AMG\\_param](#) \*param)

*Solve  $Ax = b$  by algebraic multigrid methods.*

### 9.83.1 Detailed Description

AMG method as an iterative solver.

#### Note

This file contains Level-5 (Sol) functions. It requires: [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSparseCheck.c](#), [BlaSparseCSR.c](#), [KrySPgmres.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreDataInit.c](#), and [PreMGsSolve.c](#)

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.83.2 Function Documentation

#### 9.83.2.1 fasp\_solver\_amg()

```
INT fasp_solver_amg (
    const dCSRmat * A,
    const dvector * b,
    dvector * x,
    AMG_param * param )
```

Solve  $Ax = b$  by algebraic multigrid methods.

#### Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to <a href="#">dvector</a> : the right hand side
<i>x</i>	Pointer to <a href="#">dvector</a> : the unknowns
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

04/06/2010

## Note

Refer to "Multigrid" by U. Trottenberg, C. W. Oosterlee and A. Schuller Appendix A.7 (by A. Brandt, P. Oswald and K. Stuben) Academic Press Inc., San Diego, CA, 2001.

Modified by Chensong Zhang on 07/26/2014: Add error handling for AMG setup Modified by Chensong Zhang on 02/01/2021: Add return value

Definition at line 49 of file SolAMG.c.

## 9.84 SolBLC.c File Reference

Iterative solvers for [dBLCmat](#) matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dblc\\_itsolver](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov\\_block3](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, [dCSRmat](#) \*A\_diag)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov\\_block4](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, [dCSRmat](#) \*A\_diag)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov\\_sweeping](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [INT](#) Num↔Layers, [dBLCmat](#) \*Ai, [dCSRmat](#) \*local\_A, [ivector](#) \*local\_index)  
*Solve  $Ax = b$  by standard Krylov methods.*

### 9.84.1 Detailed Description

Iterative solvers for [dBLCmat](#) matrices.

## Note

This file contains Level-5 (Sol) functions. It requires: [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSparseCSR.c](#), [KryPbcgs.c](#), [KryPgmres.c](#), [KryPminres.c](#), [KryPvfgmres.c](#), [KryPvgmres.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreBLC.c](#), and [PreDataInit.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.84.2 Function Documentation

**9.84.2.1 fasp\_solver\_dblc\_itsolver()**

```

INT fasp_solver_dblc_itsolver (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )

```

Solve  $Ax = b$  by standard Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBLCMat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

11/25/2010

Modified by Chunsheng Feng on 03/04/2016: add VBiCGstab solver  
Definition at line 54 of file SolBLC.c.

**9.84.2.2 fasp\_solver\_dblc\_krylov()**

```

INT fasp_solver_dblc_krylov (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )

```

Solve  $Ax = b$  by standard Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBLCMat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

07/18/2010

Definition at line 137 of file SolBLC.c.

**9.84.2.3 fasp\_solver\_dblc\_krylov\_block3()**

```
INT fasp_solver_dblc_krylov_block3 (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam,
    dCSRmat * A_diag )
```

Solve  $Ax = b$  by standard Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBLCMat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters for AMG solvers
<i>A_diag</i>	Digonal blocks of A

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

07/10/2014

**Warning**

Only works for 3X3 block problems!! – Xiaozhe Hu

Definition at line 189 of file SolBLC.c.

**9.84.2.4 fasp\_solver\_dblc\_krylov\_block4()**

```

INT fasp_solver_dblc_krylov_block4 (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam,
    dCSRmat * A_diag )

```

Solve  $Ax = b$  by standard Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBLCMat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters for AMG solvers
<i>A_diag</i>	Digonal blocks of A

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

07/06/2014

**Warning**

Only works for 4 by 4 block [dCSRmat](#) problems!! – Xiaozhe Hu

Definition at line 379 of file SolBLC.c.

**9.84.2.5 fasp\_solver\_dblc\_krylov\_sweeping()**

```

INT fasp_solver_dblc_krylov_sweeping (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    INT NumLayers,
    dBLCMat * Ai,
    dCSRmat * local_A,
    ivector * local_index )

```

Solve  $Ax = b$  by standard Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBLCMat</a> format
----------	---

## Parameters

<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>NumLayers</i>	Number of layers used for sweeping preconditioner
<i>Ai</i>	Pointer to the coeff matrix for the preconditioner in <a href="#">dBLCmat</a> format
<i>local_A</i>	Pointer to the local coeff matrices in the <a href="#">dCSRmat</a> format
<i>local_index</i>	Pointer to the local index in ivector format

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

05/01/2014

Definition at line 501 of file SolBLC.c.

## 9.85 SolBSR.c File Reference

Iterative solvers for [dBSRmat](#) matrices.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dbsr\\_itsolver](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for BSR matrices.*
- [INT fasp\\_solver\\_dbsr\\_krylov](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods for BSR matrices.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_diag](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_ilu](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [ILU\\_param](#) \*iluparam)  
*Solve  $Ax=b$  by ILUs preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_amg](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam)  
*Solve  $Ax=b$  by AMG preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_amg\\_nk](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, [dCSRmat](#) \*A\_nk, [dCSRmat](#) \*P\_nk, [dCSRmat](#) \*R\_nk)  
*Solve  $Ax=b$  by AMG with extra near kernel solve preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_nk\\_amg](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, const [INT](#) nk\_dim, [dvector](#) \*nk)  
*Solve  $Ax=b$  by AMG preconditioned Krylov methods with extra kernal space.*



### 9.85.1 Detailed Description

Iterative solvers for [dBSRmat](#) matrices.

#### Note

This file contains Level-5 (Sol) functions. It requires: [AuxMemory.c](#), [AuxMessage.c](#), [AuxThreads.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSmallMatInv.c](#), [BlalLUSetupBSR.c](#), [BlaSparseBSR.c](#), [BlaSparseCheck.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgmres.c](#), [KryPvfgmres.c](#), [KryPvgmres.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreBSR.c](#), and [PreDataInit.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.85.2 Function Documentation

#### 9.85.2.1 fasp\_solver\_dbsr\_itsolver()

```
INT fasp_solver_dbsr_itsolver (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )
```

Solve  $Ax=b$  by preconditioned Krylov methods for BSR matrices.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Zhiyang Zhou, Xiaozhe Hu

#### Date

10/26/2010

Modified by Chunsheng Feng on 03/04/2016: add VBiCGstab solver  
Definition at line 55 of file SolBSR.c.

#### 9.85.2.2 fasp\_solver\_dbsr\_krylov()

```
INT fasp_solver_dbsr_krylov (
    dBSRmat * A,
```

```

dvector * b,
dvector * x,
ITS_param * itparam )

```

Solve  $Ax=b$  by standard Krylov methods for BSR matrices.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Zhiyang Zhou, Xiaozhe Hu

#### Date

10/26/2010

Definition at line 139 of file SolBSR.c.

### 9.85.2.3 fasp\_solver\_dbsr\_krylov\_amg()

```

INT fasp_solver_dbsr_krylov_amg (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam )

```

Solve  $Ax=b$  by AMG preconditioned Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters of AMG

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xiaozhe Hu

## Date

03/16/2012

parameters of iterative method

Definition at line 354 of file SolBSR.c.

**9.85.2.4 fasp\_solver\_dbsr\_krylov\_amg\_nk()**

```

INT fasp_solver_dbsr_krylov_amg_nk (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam,
    dCSRmat * A_nk,
    dCSRmat * P_nk,
    dCSRmat * R_nk )

```

Solve  $Ax=b$  by AMG with extra near kernel solve preconditioned Krylov methods.

## Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters of AMG
<i>A_nk</i>	Pointer to the coeff matrix for near kernel space in <a href="#">dBSRmat</a> format
<i>P_nk</i>	Pointer to the prolongation for near kernel space in <a href="#">dBSRmat</a> format
<i>R_nk</i>	Pointer to the restriction for near kernel space in <a href="#">dBSRmat</a> format

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

05/26/2012

Definition at line 483 of file SolBSR.c.

**9.85.2.5 fasp\_solver\_dbsr\_krylov\_diag()**

```

INT fasp_solver_dbsr_krylov_diag (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )

```

Solve  $Ax=b$  by diagonal preconditioned Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou, Xiaozhe Hu

**Date**

10/26/2010

Modified by Chunsheng Feng, Zheng Li on 10/15/2012  
Definition at line 187 of file SolBSR.c.

**9.85.2.6 fasp\_solver\_dbsr\_krylov\_ilu()**

```
INT fasp_solver_dbsr_krylov_ilu (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    ILU_param * iluparam )
```

Solve  $Ax=b$  by ILUs preconditioned Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>iluparam</i>	Pointer to parameters of ILU

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Shiquang Zhang, Xiaozhe Hu

**Date**

10/26/2010

Definition at line 289 of file SolBSR.c.

### 9.85.2.7 fasp\_solver\_dbsr\_krylov\_nk\_amg()

```
INT fasp_solver_dbsr_krylov_nk_amg (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam,
    const INT nk_dim,
    dvector * nk )
```

Solve  $Ax=b$  by AMG preconditioned Krylov methods with extra kernal space.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters of AMG
<i>nk_dim</i>	Dimension of the near kernel spaces
<i>nk</i>	Pointer to the near kernal spaces

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xiaozhe Hu

#### Date

05/27/2012

parameters of iterative method  
Definition at line 640 of file SolBSR.c.

## 9.86 SolCSR.c File Reference

Iterative solvers for [dCSRmat](#) matrices.

```
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dcsr\\_itsolver](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.*
- [INT fasp\\_solver\\_dcsr\\_itsolver\\_s](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods with safe-net for CSR matrices.*

- `INT fasp_solver_dcsr_krylov (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam)`  
*Solve  $Ax=b$  by standard Krylov methods for CSR matrices.*
- `INT fasp_solver_dcsr_krylov_s (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam)`  
*Solve  $Ax=b$  by standard Krylov methods with safe-net for CSR matrices.*
- `INT fasp_solver_dcsr_krylov_diag (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam)`  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*
- `INT fasp_solver_dcsr_krylov_swz (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam, SWZ_param *schparam)`  
*Solve  $Ax=b$  by overlapping Schwarz Krylov methods.*
- `INT fasp_solver_dcsr_krylov_amg (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam, AMG_param *amgparam)`  
*Solve  $Ax=b$  by AMG preconditioned Krylov methods.*
- `INT fasp_solver_dcsr_krylov_ilu (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam, ILU_param *iluparam)`  
*Solve  $Ax=b$  by ILUs preconditioned Krylov methods.*
- `INT fasp_solver_dcsr_krylov_ilu_M (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam, ILU_param *iluparam, dCSRmat *M)`  
*Solve  $Ax=b$  by ILUs preconditioned Krylov methods: ILU of M as preconditioner.*
- `INT fasp_solver_dcsr_krylov_amg_nk (dCSRmat *A, dvector *b, dvector *x, ITS_param *itparam, AMG_param *amgparam, dCSRmat *A_nk, dCSRmat *P_nk, dCSRmat *R_nk)`  
*Solve  $Ax=b$  by AMG preconditioned Krylov methods with an extra near kernel solve.*

### 9.86.1 Detailed Description

Iterative solvers for `dCSRmat` matrices.

#### Note

This file contains Level-5 (Sol) functions. It requires: `AuxMemory.c`, `AuxMessage.c`, `AuxParam.c`, `AuxTiming.c`, `AuxVector.c`, `BlaILUSetupCSR.c`, `BlaSchwarzSetup.c`, `BlaSparseCheck.c`, `BlaSparseCSR.c`, `KryPbcgs.c`, `KryPcg.c`, `KryPgcr.c`, `KryPgmres.c`, `KryPminres.c`, `KryPvfgmres.c`, `KryPvgmres.c`, `PreAMGSetupRS.c`, `PreAMGSetupSA.c`, `PreAMGSetupUA.c`, `PreCSR.c`, and `PreDataInit.c`

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.86.2 Function Documentation

#### 9.86.2.1 fasp\_solver\_dcsr\_itsolver()

```
INT fasp_solver_dcsr_itsolver (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )
```

Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.

#### Note

This is an abstract interface for iterative methods.

## Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

09/25/2009

Definition at line 56 of file SolCSR.c.

## 9.86.2.2 fasp\_solver\_dcsr\_itsolver\_s()

```

INT fasp_solver_dcsr_itsolver_s (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )

```

Solve  $Ax=b$  by preconditioned Krylov methods with safe-net for CSR matrices.

## Note

This is an abstract interface for iterative methods.

## Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

10/21/2017

Definition at line 158 of file SolCSR.c.

**9.86.2.3 fasp\_solver\_dcsr\_krylov()**

```

INT fasp_solver_dcsr_krylov (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )

```

Solve  $Ax=b$  by standard Krylov methods for CSR matrices.

## Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang, Shiquan Zhang

## Date

09/25/2009

Definition at line 245 of file SolCSR.c.

**9.86.2.4 fasp\_solver\_dcsr\_krylov\_amg()**

```

INT fasp_solver_dcsr_krylov_amg (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam )

```

Solve  $Ax=b$  by AMG preconditioned Krylov methods.

## Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters for AMG methods



**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

09/25/2009

Definition at line 483 of file SolCSR.c.

**9.86.2.5 fasp\_solver\_dcsr\_krylov\_amg\_nk()**

```

INT fasp_solver_dcsr_krylov_amg_nk (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    AMG_param * amgparam,
    dCSRmat * A_nk,
    dCSRmat * P_nk,
    dCSRmat * R_nk )

```

Solve  $Ax=b$  by AMG preconditioned Krylov methods with an extra near kernel solve.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>amgparam</i>	Pointer to parameters for AMG methods
<i>A_nk</i>	Pointer to the coeff matrix of near kernel space in <a href="#">dCSRmat</a> format
<i>P_nk</i>	Pointer to the prolongation of near kernel space in <a href="#">dCSRmat</a> format
<i>R_nk</i>	Pointer to the restriction of near kernel space in <a href="#">dCSRmat</a> format

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

05/26/2014

Definition at line 753 of file SolCSR.c.

### 9.86.2.6 fasp\_solver\_dcsr\_krylov\_diag()

```
INT fasp_solver_dcsr_krylov_diag (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )
```

Solve  $Ax=b$  by diagonal preconditioned Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang, Shiquan Zhang

#### Date

09/25/2009

Definition at line 343 of file SolCSR.c.

### 9.86.2.7 fasp\_solver\_dcsr\_krylov\_ilu()

```
INT fasp_solver_dcsr_krylov_ilu (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    ILU_param * iluparam )
```

Solve  $Ax=b$  by ILUs preconditioned Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>iluparam</i>	Pointer to parameters for ILU

#### Returns

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang, Shiquan Zhang

**Date**

09/25/2009

Definition at line 587 of file SolCSR.c.

**9.86.2.8 fasp\_solver\_dcsr\_krylov\_ilu\_M()**

```

INT fasp_solver_dcsr_krylov_ilu_M (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    ILU_param * iluparam,
    dCSRmat * M )

```

Solve  $Ax=b$  by ILUs preconditioned Krylov methods: ILU of  $M$  as preconditioner.**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>iluparam</i>	Pointer to parameters for ILU
<i>M</i>	Pointer to the preconditioning matrix in <a href="#">dCSRmat</a> format

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

09/25/2009

**Note**

This function is specially designed for reservoir simulation. Have not been tested in any other places.

Definition at line 670 of file SolCSR.c.

**9.86.2.9 fasp\_solver\_dcsr\_krylov\_s()**

```

INT fasp_solver_dcsr_krylov_s (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )

```

Solve  $Ax=b$  by standard Krylov methods with safe-net for CSR matrices.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Chensong Zhang

**Date**

10/22/2017

Definition at line 294 of file SolCSR.c.

**9.86.2.10 fasp\_solver\_dcsr\_krylov\_swz()**

```

INT fasp_solver_dcsr_krylov_swz (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    SWZ_param * schparam )

```

Solve  $Ax=b$  by overlapping Schwarz Krylov methods.

**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dCSRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>schparam</i>	Pointer to parameters for Schwarz methods

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

03/21/2011

Modified by Chensong on 07/02/2012: change interface  
Definition at line 405 of file SolCSR.c.

## 9.87 SolFAMG.c File Reference

Full AMG method as an iterative solver.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- void [fasp\\_solver\\_famg](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [AMG\\_param](#) \*param)  
*Solve  $Ax=b$  by full AMG.*

### 9.87.1 Detailed Description

Full AMG method as an iterative solver.

Note

This file contains Level-5 (Sol) functions. It requires: [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSparseCheck.c](#), [BlaSparseCSR.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreDataInit.c](#), and [PreMGSolve.c](#)

---

Copyright (C) 2009–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.87.2 Function Documentation

#### 9.87.2.1 [fasp\\_solver\\_famg\(\)](#)

```
INT fasp_solver_famg (
    const dCSRmat * A,
    const dvector * b,
    dvector * x,
    AMG\_param * param )
```

Solve  $Ax=b$  by full AMG.

Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to <a href="#">dvector</a> : the right hand side
<i>x</i>	Pointer to <a href="#">dvector</a> : the unknowns
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

Author

Xiaozhe Hu

Date

02/27/2011

Modified by Chensong Zhang on 05/05/2013: Remove error handling for AMG setup  
Definition at line 41 of file SolFAMG.c.

## 9.88 SolGMGPoisson.c File Reference

GMG method as an iterative solver for Poisson Problem.

```
#include <time.h>
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "PreGMG.inl"
```

### Functions

- **INT fasp\_poisson\_gmg1d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method.*
- **INT fasp\_poisson\_gmg2d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method.*
- **INT fasp\_poisson\_gmg3d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** nz, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method.*
- void **fasp\_poisson\_fgmg1d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method (FMG)*
- void **fasp\_poisson\_fgmg2d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method (FMG)*
- void **fasp\_poisson\_fgmg3d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** nz, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method (FMG)*
- **INT fasp\_poisson\_gmgcg1d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)*
- **INT fasp\_poisson\_gmgcg2d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)*
- **INT fasp\_poisson\_gmgcg3d** (**REAL** \*u, **REAL** \*b, const **INT** nx, const **INT** ny, const **INT** nz, const **INT** maxlevel, const **REAL** rtol, const **SHORT** prtvl)
 

*Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)*

### 9.88.1 Detailed Description

GMG method as an iterative solver for Poisson Problem.

Note

This file contains Level-5 (Sol) functions. It requires: [AuxArray.c](#), [AuxMessage.c](#), and [AuxTiming.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.88.2 Function Documentation

**9.88.2.1 fasp\_poisson\_fgm1d()**

```
void fasp_poisson_fgm1d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )
```

Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method (FMG)

**Parameters**

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 442 of file SolGMGPoisson.c.

**9.88.2.2 fasp\_poisson\_fgm2d()**

```
void fasp_poisson_fgm2d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )
```

Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method (FMG)

**Parameters**

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in Y direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 536 of file SolGMGPoisson.c.

**9.88.2.3 fasp\_poisson\_fgmg3d()**

```
void fasp_poisson_fgmg3d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT nz,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )
```

Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method (FMG)**Parameters**

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	NUmber of grids in y direction
<i>nz</i>	NUmber of grids in z direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 644 of file SolGMGPoisson.c.

**9.88.2.4 fasp\_poisson\_gmg1d()**

```
INT fasp_poisson_gmg1d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )
```



Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method.

## Parameters

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Ziteng Wang, Chensong Zhang

## Date

06/07/2013

Definition at line 48 of file SolGMGPoisson.c.

## 9.88.2.5 fasp\_poisson\_gmg2d()

```

INT fasp_poisson_gmg2d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )

```

Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method.

## Parameters

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

## Returns

Iteration number if converges; ERROR otherwise.

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 172 of file SolGMGPoisson.c.

**9.88.2.6 fasp\_poisson\_gmg3d()**

```

INT fasp_poisson_gmg3d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT nz,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )

```

Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method.**Parameters**

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>nz</i>	Number of grids in z direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 308 of file SolGMGPoisson.c.

**9.88.2.7 fasp\_poisson\_gmgcg1d()**

```

INT fasp_poisson_gmgcg1d (
    REAL * u,

```

```

    REAL * b,
    const INT nx,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )

```

Solve  $Ax=b$  of Poisson 1D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)

#### Parameters

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Ziteng Wang, Chensong Zhang

#### Date

06/07/2013

Definition at line 754 of file SolGMGPoisson.c.

### 9.88.2.8 fasp\_poisson\_gmgcg2d()

```

INT fasp_poisson_gmgcg2d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )

```

Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)

#### Parameters

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 849 of file SolGMGPoisson.c.

**9.88.2.9 fasp\_poisson\_gmgcg3d()**

```
INT fasp_poisson_gmgcg3d (
    REAL * u,
    REAL * b,
    const INT nx,
    const INT ny,
    const INT nz,
    const INT maxlevel,
    const REAL rtol,
    const SHORT prtlvl )
```

Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)

**Parameters**

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>nz</i>	Number of grids in z direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Ziteng Wang, Chensong Zhang

**Date**

06/07/2013

Definition at line 959 of file SolGMGPoisson.c.

## 9.89 SolMatFree.c File Reference

Iterative solvers using MatFree spmv operations.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp_block.h"
#include "KryUtil.inl"
#include "BlaSpmvMatFree.inl"
```

### Functions

- [INT fasp\\_solver\\_itsolver](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.*
- [INT fasp\\_solver\\_krylov](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods – without preconditioner.*
- void [fasp\\_solver\\_matfree\\_init](#) ([INT](#) matrix\_format, [mxv\\_matfree](#) \*mf, void \*A)  
*Initialize MatFree (or non-specified format) itsolvers.*

### 9.89.1 Detailed Description

Iterative solvers using MatFree spmv operations.

#### Note

This file contains Level-5 (Sol) functions. It requires: [AuxMessage.c](#), [AuxTiming.c](#), [BlaSpmvBLC.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [BlaSpmvCSRL.c](#), [BlaSpmvSTR.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgcg.c](#), [KryPgmres.c](#), [KryPminres.c](#), [KryPvfgmres.c](#), and [KryPvgmres.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.89.2 Function Documentation

#### 9.89.2.1 fasp\_solver\_itsolver()

```
INT fasp_solver_itsolver (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )
```

Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.

#### Note

This is an abstract interface for iterative methods.

#### Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> MatFree spmv operation
<i>b</i>	Pointer to the right hand side in dvector format

## Parameters

<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

09/25/2009

Modified by Feiteng Huang on 09/19/2012: matrix free  
Definition at line 58 of file SolMatFree.c.

## 9.89.2.2 fasp\_solver\_krylov()

```
INT fasp_solver_krylov (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    ITS_param * itparam )
```

Solve  $Ax=b$  by standard Krylov methods – without preconditioner.

## Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> MatFree spmv operation
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

## Returns

Number of iterations if succeed

## Author

Chensong Zhang, Shiquan Zhang

## Date

09/25/2009

Modified by Feiteng Huang on 09/20/2012: matrix free  
Definition at line 154 of file SolMatFree.c.

### 9.89.2.3 fasp\_solver\_matfree\_init()

```
void fasp_solver_matfree_init (
    INT matrix_format,
    mxv_matfree * mf,
    void * A )
```

Initialize MatFree (or non-specified format) itsolvers.

#### Parameters

<i>matrix_format</i>	matrix format
<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> MatFree spmv operation
<i>A</i>	void pointer to the coefficient matrix

#### Author

Feiteng Huang

#### Date

09/18/2012

Modified by Chensong Zhang on 05/10/2013: Change interface of mat-free mv  
Definition at line 201 of file SolMatFree.c.

## 9.90 SolSTR.c File Reference

Iterative solvers for [dSTRmat](#) matrices.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "KryUtil.inl"
```

### Functions

- [INT fasp\\_solver\\_dstr\\_itsolver](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov\\_diag](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov\\_ilu](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [ILU\\_param](#) \*iluparam)  
*Solve  $Ax=b$  by structured ILU preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov\\_blockgs](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [ITS\\_param](#) \*itparam, [ivector](#) \*neigh, [ivector](#) \*order)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*



## 9.90.1 Detailed Description

Iterative solvers for [dSTRmat](#) matrices.

### Note

This file contains Level-5 (Sol) functions. It requires: [AuxArray.c](#), [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSmallMatInv.c](#), [BlalLUSetupSTR.c](#), [BlaSparseSTR.c](#), [ItrSmootherSTR.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgmres.c](#), [KryPvgmres.c](#), and [PreSTR.c](#)

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

## 9.90.2 Function Documentation

### 9.90.2.1 fasp\_solver\_dstr\_itsolver()

```
INT fasp_solver_dstr_itsolver (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    ITS_param * itparam )
```

Solve  $Ax=b$  by standard Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dSTRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

09/25/2009

Modified by Chunsheng Feng on 03/04/2016: add VBiCGstab solver  
Definition at line 51 of file SolSTR.c.

### 9.90.2.2 fasp\_solver\_dstr\_krylov()

```
INT fasp_solver_dstr_krylov (
    dSTRmat * A,
```

```

dvector * b,
dvector * x,
ITS_param * itparam )

```

Solve  $Ax=b$  by standard Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dSTRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Zhiyang Zhou

#### Date

04/25/2010

Definition at line 131 of file SolSTR.c.

### 9.90.2.3 fasp\_solver\_dstr\_krylov\_blockgs()

```

INT fasp_solver_dstr_krylov_blockgs (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    ivector * neigh,
    ivector * order )

```

Solve  $Ax=b$  by diagonal preconditioned Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <a href="#">dSTRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>neigh</i>	Pointer to neighbor vector
<i>order</i>	Pointer to solver ordering

#### Returns

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

10/10/2010

Definition at line 334 of file SolSTR.c.

**9.90.2.4 fasp\_solver\_dstr\_krylov\_diag()**

```
INT fasp_solver_dstr_krylov_diag (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam )
```

Solve  $Ax=b$  by diagonal preconditioned Krylov methods.**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dSTRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Zhiyang Zhou

**Date**

4/23/2010

Definition at line 177 of file SolSTR.c.

**9.90.2.5 fasp\_solver\_dstr\_krylov\_ilu()**

```
INT fasp_solver_dstr_krylov_ilu (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    ITS_param * itparam,
    ILU_param * iluparam )
```

Solve  $Ax=b$  by structured ILU preconditioned Krylov methods.**Parameters**

<i>A</i>	Pointer to the coeff matrix in <a href="#">dSTRmat</a> format
<i>b</i>	Pointer to the right hand side in dvector format

## Parameters

<i>x</i>	Pointer to the approx solution in dvector format
<i>itparam</i>	Pointer to parameters for iterative solvers
<i>iluparam</i>	Pointer to parameters for ILU

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

05/01/2010

Definition at line 241 of file SolSTR.c.

## 9.91 SolWrapper.c File Reference

Wrappers for accessing functions by advanced users.

```
#include "fasp.h"
#include "fasp_block.h"
#include "fasp_functs.h"
```

### Functions

- void `fasp_fwrapper_dcsr_pardiso_` (INT \*n, INT \*nnz, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, INT \*ptrlvl)  
*Solve  $Ax=b$  by the Pardiso direct solver.*
- void `fasp_fwrapper_dcsr_amg_` (INT \*n, INT \*nnz, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, REAL \*tol, INT \*maxit, INT \*ptrlvl)  
*Solve  $Ax=b$  by Ruge and Stuben's classic AMG.*
- void `fasp_fwrapper_dcsr_krylov_ilu_` (INT \*n, INT \*nnz, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, REAL \*tol, INT \*maxit, INT \*ptrlvl)  
*Solve  $Ax=b$  by Krylov method preconditioned by ILUk.*
- void `fasp_fwrapper_dcsr_krylov_amg_` (INT \*n, INT \*nnz, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, REAL \*tol, INT \*maxit, INT \*ptrlvl)  
*Solve  $Ax=b$  by Krylov method preconditioned by classic AMG.*
- void `fasp_fwrapper_dbsr_krylov_ilu_` (INT \*n, INT \*nnz, INT \*nb, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, REAL \*tol, INT \*maxit, INT \*ptrlvl)
- void `fasp_fwrapper_dbsr_krylov_amg_` (INT \*n, INT \*nnz, INT \*nb, INT \*ia, INT \*ja, REAL \*a, REAL \*b, REAL \*u, REAL \*tol, INT \*maxit, INT \*ptrlvl)

### 9.91.1 Detailed Description

Wrappers for accessing functions by advanced users.

**Note**

This file contains Level-5 (Sol) functions. It requires: [AuxParam.c](#), [BlaFormat.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [SolAMG.c](#), [SolBSR.c](#), and [SolCSR.c](#)

IMPORTANT: The wrappers DO NOT change the original matrix data. Users should shift the matrix indices in order to make the IA and JA to start from 0 instead of 1.

---

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

**9.91.2 Function Documentation****9.91.2.1 fasp\_fwrapper\_dcsr\_amg\_()**

```
void fasp_fwrapper_dcsr_amg_ (
    INT * n,
    INT * nnz,
    INT * ia,
    INT * ja,
    REAL * a,
    REAL * b,
    REAL * u,
    REAL * tol,
    INT * maxit,
    INT * ptrlvl )
```

Solve  $Ax=b$  by Ruge and Stuben's classic AMG.

**Parameters**

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>ia</i>	IA of A in CSR format
<i>ja</i>	JA of A in CSR format
<i>a</i>	VAL of A in CSR format
<i>b</i>	RHS vector
<i>u</i>	Solution vector
<i>tol</i>	Tolerance for iterative solvers
<i>maxit</i>	Max number of iterations
<i>ptrlvl</i>	Print level for iterative solvers

**Author**

Chensong Zhang

**Date**

09/16/2010

Definition at line 90 of file SolWrapper.c.

### 9.91.2.2 fasp\_fwrapper\_dcsr\_krylov\_amg\_()

```
void fasp_fwrapper_dcsr_krylov_amg_ (
    INT * n,
    INT * nnz,
    INT * ia,
    INT * ja,
    REAL * a,
    REAL * b,
    REAL * u,
    REAL * tol,
    INT * maxit,
    INT * ptrlvl )
```

Solve  $Ax=b$  by Krylov method preconditioned by classic AMG.

#### Parameters

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>ia</i>	IA of A in CSR format
<i>ja</i>	JA of A in CSR format
<i>a</i>	VAL of A in CSR format
<i>b</i>	RHS vector
<i>u</i>	Solution vector
<i>tol</i>	Tolerance for iterative solvers
<i>maxit</i>	Max number of iterations
<i>ptrlvl</i>	Print level for iterative solvers

#### Author

Chensong Zhang

#### Date

09/16/2010

Definition at line 203 of file SolWrapper.c.

### 9.91.2.3 fasp\_fwrapper\_dcsr\_krylov\_ilu\_()

```
void fasp_fwrapper_dcsr_krylov_ilu_ (
    INT * n,
    INT * nnz,
    INT * ia,
    INT * ja,
    REAL * a,
    REAL * b,
    REAL * u,
    REAL * tol,
    INT * maxit,
    INT * ptrlvl )
```

Solve  $Ax=b$  by Krylov method preconditioned by ILUK.

## Parameters

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>ia</i>	IA of A in CSR format
<i>ja</i>	JA of A in CSR format
<i>a</i>	VAL of A in CSR format
<i>b</i>	RHS vector
<i>u</i>	Solution vector
<i>tol</i>	Tolerance for iterative solvers
<i>maxit</i>	Max number of iterations
<i>ptrlvl</i>	Print level for iterative solvers

## Author

Chensong Zhang

## Date

03/24/2018

Definition at line 143 of file SolWrapper.c.

**9.91.2.4 fasp\_fwrapper\_dcsr\_pardiso\_()**

```
void fasp_fwrapper_dcsr_pardiso_ (
    INT * n,
    INT * nnz,
    INT * ia,
    INT * ja,
    REAL * a,
    REAL * b,
    REAL * u,
    INT * ptrlvl )
```

Solve  $Ax=b$  by the Pardiso direct solver.

## Parameters

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>ia</i>	IA of A in CSR format
<i>ja</i>	JA of A in CSR format
<i>a</i>	VAL of A in CSR format
<i>b</i>	RHS vector
<i>u</i>	Solution vector
<i>ptrlvl</i>	Print level for iterative solvers

## Author

Chensong Zhang

Date

01/09/2020

Definition at line 45 of file SolWrapper.c.

## 9.92 XtrMumps.c File Reference

Interface to MUMPS direct solvers.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Macros

- #define [ICNTL](#)(I) icntl[(I)-1]

### Functions

- int [fasp\\_solver\\_mumps](#) (dCSRmat \*ptrA, dvector \*b, dvector \*u, const [SHORT](#) prtlvl)  
*Solve  $Ax=b$  by MUMPS directly.*
- int [fasp\\_solver\\_mumps\\_steps](#) (dCSRmat \*ptrA, dvector \*b, dvector \*u, [Mumps\\_data](#) \*mumps)  
*Solve  $Ax=b$  by MUMPS in three steps.*

### 9.92.1 Detailed Description

Interface to MUMPS direct solvers.

Reference for MUMPS: <http://mumps.enseeiht.fr/>  
Copyright (C) 2013–2020 by the FASP team. All rights reserved.

---

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.92.2 Macro Definition Documentation

#### 9.92.2.1 ICNTL

```
#define ICNTL(
    I ) icntl[(I)-1]
macro s.t. indices match documentation
Definition at line 23 of file XtrMumps.c.
```

### 9.92.3 Function Documentation

#### 9.92.3.1 fasp\_solver\_mumps()

```
int fasp_solver_mumps (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    const SHORT prtlvl )
```



Solve  $Ax=b$  by MUMPS directly.

**Parameters**

<i>ptrA</i>	Pointer to a <a href="#">dCSRmat</a> matrix
<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>prtlvl</i>	Output level

**Author**

Chunsheng Feng

**Date**

02/27/2013

Modified by Chensong Zhang on 02/27/2013 for new FASP function names.  
Definition at line 45 of file XtrMumps.c.

**9.92.3.2 fasp\_solver\_mumps\_steps()**

```
int fasp_solver_mumps_steps (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    Mumps_data * mumps )
```

Solve  $Ax=b$  by MUMPS in three steps.

**Parameters**

<i>ptrA</i>	Pointer to a <a href="#">dCSRmat</a> matrix
<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>mumps</i>	Pointer to MUMPS data

**Author**

Chunsheng Feng

**Date**

02/27/2013

Modified by Chensong Zhang on 02/27/2013 for new FASP function names. Modified by Zheng Li on 10/10/2014 to adjust input parameters. Modified by Chunsheng Feng on 08/11/2017 for debug information.  
Definition at line 176 of file XtrMumps.c.

**9.93 XtrPardiso.c File Reference**

Interface to Intel MKL PARDISO direct solvers.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- `INT fasp_solver_pardiso (dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)`

*Solve  $Ax=b$  by PARDISO directly.*

### 9.93.1 Detailed Description

Interface to Intel MKL PARDISO direct solvers.

Reference for Intel MKL PARDISO: <https://software.intel.com/en-us/node/470282>

Copyright (C) 2015–Present by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.93.2 Function Documentation

#### 9.93.2.1 fasp\_solver\_pardiso()

```
INT fasp_solver_pardiso (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    const SHORT prtlvl )
```

Solve  $Ax=b$  by PARDISO directly.

#### Parameters

<i>ptrA</i>	Pointer to a <code>dCSRmat</code> matrix
<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>prtlvl</i>	Output level

#### Author

Hongxuan Zhang

#### Date

11/28/2015

#### Note

Each row of A should be in ascending order w.r.t. column indices.

Definition at line 45 of file XtrPardiso.c.

## 9.94 XtrSamg.c File Reference

Interface to SAMG solvers.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

## Functions

- void `dvector2SAMGInput` (`dvector` \*vec, char \*filename)  
Write a dvector to disk file in SAMG format (coordinate format)
- `INT dCSRmat2SAMGInput` (`dCSRmat` \*A, char \*filefrm, char \*fileamg)  
Write SAMG Input data from a sparse matrix of CSR format.

### 9.94.1 Detailed Description

Interface to SAMG solvers.

Reference for SAMG: <http://www.scai.fraunhofer.de/geschaeftsfelder/nuso/produkte/samg.html>

#### Warning

This interface has *only* been tested for SAMG24a1 (2010 version)!

---

Copyright (C) 2010–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.94.2 Function Documentation

#### 9.94.2.1 dCSRmat2SAMGInput()

```
INT dCSRmat2SAMGInput (
    dCSRmat * A,
    char * filefrm,
    char * fileamg )
```

Write SAMG Input data from a sparse matrix of CSR format.

#### Parameters

<i>A</i>	Pointer to the <code>dCSRmat</code> matrix
<i>filefrm</i>	Name of the .frm file
<i>fileamg</i>	Name of the .amg file

#### Author

Zhiyang Zhou

#### Date

2010/08/25

Definition at line 65 of file XtrSamg.c.

#### 9.94.2.2 dvector2SAMGInput()

```
void dvector2SAMGInput (
    dvector * vec,
    char * filename )
```

Write a dvector to disk file in SAMG format (coordinate format)

## Parameters

<i>vec</i>	Pointer to the dvector
<i>filename</i>	File name for input

## Author

Zhiyang Zhou

## Date

08/25/2010

Definition at line 36 of file XtrSamg.c.

## 9.95 XtrSuperlu.c File Reference

Interface to SuperLU direct solvers.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

### Functions

- int [fasp\\_solver\\_superlu](#) (dCSRmat \*ptrA, dvector \*b, dvector \*u, const SHORT prtlvl)  
*Solve  $Au=b$  by SuperLU.*

#### 9.95.1 Detailed Description

Interface to SuperLU direct solvers.

Reference for SuperLU: <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>  
Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

#### 9.95.2 Function Documentation

##### 9.95.2.1 fasp\_solver\_superlu()

```
int fasp_solver_superlu (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    const SHORT prtlvl )
```

Solve  $Au=b$  by SuperLU.

## Parameters

<i>ptrA</i>	Pointer to a <a href="#">dCSRmat</a> matrix
-------------	---

**Parameters**

<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>prtlvl</i>	Output level

**Author**

Xiaozhe Hu

**Date**

11/05/2009

Modified by Chensong Zhang on 02/27/2013 for new FASP function names.

**Note**

Factorization and solution are combined together!!! Not efficient!!!

Definition at line 47 of file XtrSuperlu.c.

## 9.96 XtrUmfpack.c File Reference

Interface to UMFPACK direct solvers.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
```

**Functions**

- `INT fasp_solver_umfpack (dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)`  
*Solve  $Au=b$  by UMFPack.*

### 9.96.1 Detailed Description

Interface to UMFPACK direct solvers.

Reference for SuiteSparse: <http://faculty.cse.tamu.edu/davis/suitesparse.html>

Copyright (C) 2009–2020 by the FASP team. All rights reserved.

Released under the terms of the GNU Lesser General Public License 3.0 or later.

### 9.96.2 Function Documentation

#### 9.96.2.1 fasp\_solver\_umfpack()

```
INT fasp_solver_umfpack (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    const SHORT prtlvl )
```

Solve  $Au=b$  by UMFPack.

## Parameters

<i>ptrA</i>	Pointer to a <a href="#">dCSRmat</a> matrix
<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>prtlvl</i>	Output level

## Author

Chensong Zhang

## Date

05/20/2010

Modified by Chensong Zhang on 02/27/2013 for new FASP function names.  
Definition at line 43 of file XtrUmfpack.c.





# Index

\_\_FASPBLOCK\_HEADER\_\_  
    fasp\_block.h, [273](#)  
\_\_FASPGRID\_HEADER\_\_  
    fasp\_grid.h, [297](#)  
\_\_FASP\_HEADER\_\_  
    fasp.h, [268](#)

A  
    precond\_data\_sweeping, [64](#)  
A\_diag  
    precond\_data\_blc, [59](#)  
Ablc  
    precond\_data\_blc, [59](#)  
ABS  
    fasp.h, [268](#)  
Ai  
    precond\_data\_sweeping, [64](#)  
AMG\_aggregation\_type  
    input\_param, [40](#)  
AMG\_aggressive\_level  
    input\_param, [40](#)  
AMG\_aggressive\_path  
    input\_param, [40](#)  
AMG\_amli\_degree  
    input\_param, [41](#)  
AMG\_coarse\_dof  
    input\_param, [41](#)  
AMG\_coarse\_scaling  
    input\_param, [41](#)  
AMG\_coarse\_solver  
    input\_param, [41](#)  
AMG\_coarsening\_type  
    input\_param, [41](#)  
AMG\_cycle\_type  
    input\_param, [42](#)  
AMG\_data, [17](#)  
AMG\_data\_bsr, [18](#)  
AMG\_ILU\_levels  
    input\_param, [42](#)  
AMG\_interpolation\_type  
    input\_param, [42](#)  
AMG\_levels  
    input\_param, [42](#)  
AMG\_max\_aggregation  
    input\_param, [42](#)  
AMG\_max\_row\_sum

    input\_param, [43](#)  
AMG\_maxit  
    input\_param, [43](#)  
AMG\_nl\_amli\_krylov\_type  
    input\_param, [43](#)  
AMG\_pair\_number  
    input\_param, [43](#)  
AMG\_param, [20](#)  
AMG\_polynomial\_degree  
    input\_param, [43](#)  
AMG\_postsmooth\_iter  
    input\_param, [44](#)  
AMG\_presmooth\_iter  
    input\_param, [44](#)  
AMG\_quality\_bound  
    input\_param, [44](#)  
AMG\_relaxation  
    input\_param, [44](#)  
AMG\_smooth\_filter  
    input\_param, [44](#)  
AMG\_smooth\_order  
    input\_param, [45](#)  
AMG\_smooth\_restriction  
    input\_param, [45](#)  
AMG\_smoother  
    input\_param, [45](#)  
AMG\_strong\_coupled  
    input\_param, [45](#)  
AMG\_strong\_threshold  
    input\_param, [45](#)  
AMG\_SWZ\_levels  
    input\_param, [46](#)  
AMG\_tentative\_smooth  
    input\_param, [46](#)  
AMG\_tol  
    input\_param, [46](#)  
AMG\_truncation\_threshold  
    input\_param, [46](#)  
AMG\_type  
    input\_param, [46](#)  
amgparam  
    precond\_data\_blc, [59](#)  
AMLI\_CYCLE  
    fasp\_const.h, [278](#)  
ASCEND

- fasp\_const.h, 278
- AuxArray.c, 71
  - fasp\_darray\_cp, 72
  - fasp\_darray\_set, 72
  - fasp\_iarray\_cp, 73
  - fasp\_iarray\_set, 73
- AuxConvert.c, 74
  - fasp\_aux\_bbyteToldouble, 74
  - fasp\_aux\_change\_endian4, 74
  - fasp\_aux\_change\_endian8, 75
- AuxGivens.c, 75
  - fasp\_aux\_givens, 76
- AuxGraphics.c, 76
  - fasp\_dbsr\_plot, 77
  - fasp\_dbsr\_subplot, 78
  - fasp\_dcsr\_plot, 78
  - fasp\_dcsr\_subplot, 79
  - fasp\_grid2d\_plot, 79
- AuxInput.c, 80
  - fasp\_param\_check, 80
  - fasp\_param\_input, 81
- AuxMemory.c, 81
  - fasp\_mem\_free, 82
  - fasp\_mem\_iludata\_check, 82
  - fasp\_mem\_realloc, 83
  - fasp\_mem\_usage, 83
- AuxMessage.c, 84
  - fasp\_amgcomplexity, 84
  - fasp\_amgcomplexity\_bsr, 85
  - fasp\_chkerr, 85
  - fasp\_cputime, 85
  - fasp\_itinfo, 86
  - fasp\_message, 86
- AuxParam.c, 87
  - fasp\_param\_amg\_init, 88
  - fasp\_param\_amg\_print, 89
  - fasp\_param\_amg\_set, 89
  - fasp\_param\_amg\_to\_prec, 89
  - fasp\_param\_amg\_to\_precbsr, 91
  - fasp\_param\_ilu\_init, 91
  - fasp\_param\_ilu\_print, 92
  - fasp\_param\_ilu\_set, 92
  - fasp\_param\_init, 92
  - fasp\_param\_input\_init, 93
  - fasp\_param\_prec\_to\_amg, 93
  - fasp\_param\_precbsr\_to\_amg, 94
  - fasp\_param\_set, 94
  - fasp\_param\_solver\_init, 95
  - fasp\_param\_solver\_print, 95
  - fasp\_param\_solver\_set, 95
  - fasp\_param\_swz\_init, 96
  - fasp\_param\_swz\_print, 96
  - fasp\_param\_swz\_set, 97
- AuxSort.c, 97
  - fasp\_aux\_BiSearch, 98
  - fasp\_aux\_dQuickSort, 98
  - fasp\_aux\_dQuickSortIndex, 100
  - fasp\_aux\_iQuickSort, 100
  - fasp\_aux\_iQuickSortIndex, 101
  - fasp\_aux\_merge, 102
  - fasp\_aux\_msort, 102
  - fasp\_aux\_unique, 103
- AuxThreads.c, 103
  - fasp\_get\_start\_end, 104
  - fasp\_set\_gs\_threads, 104
  - THDs\_AMG\_GS, 105
  - THDs\_CPR\_gGS, 105
  - THDs\_CPR\_IGS, 105
- AuxTiming.c, 105
  - fasp\_gettime, 106
- AuxVector.c, 106
  - fasp\_dvec\_alloc, 107
  - fasp\_dvec\_cp, 108
  - fasp\_dvec\_create, 108
  - fasp\_dvec\_free, 108
  - fasp\_dvec\_isnan, 109
  - fasp\_dvec\_maxdiff, 109
  - fasp\_dvec\_rand, 110
  - fasp\_dvec\_set, 111
  - fasp\_dvec\_symdiagscale, 111
  - fasp\_ivec\_alloc, 112
  - fasp\_ivec\_create, 112
  - fasp\_ivec\_free, 112
  - fasp\_ivec\_set, 113
- BIGREAL
  - fasp\_const.h, 278
- BlaArray.c, 113
  - fasp\_blas\_darray\_ax, 115
  - fasp\_blas\_darray\_axpby, 115
  - fasp\_blas\_darray\_axpy, 116
  - fasp\_blas\_darray\_axpy\_nc2, 116
  - fasp\_blas\_darray\_axpy\_nc3, 117
  - fasp\_blas\_darray\_axpy\_nc5, 117
  - fasp\_blas\_darray\_axpy\_nc7, 118
  - fasp\_blas\_darray\_axpyz, 118
  - fasp\_blas\_darray\_axpyz\_nc2, 119
  - fasp\_blas\_darray\_axpyz\_nc3, 119
  - fasp\_blas\_darray\_axpyz\_nc5, 120
  - fasp\_blas\_darray\_axpyz\_nc7, 120
  - fasp\_blas\_darray\_dotprod, 121
  - fasp\_blas\_darray\_norm1, 121
  - fasp\_blas\_darray\_norm2, 122
  - fasp\_blas\_darray\_norminf, 122
- BlaEigen.c, 123
  - fasp\_dcsr\_maxeig, 123
- BlaFormat.c, 124
  - fasp\_format\_dbld\_csr, 125

- fasp\_format\_dbsr\_dcoo, 125
- fasp\_format\_dbsr\_dcsr, 125
- fasp\_format\_dcoo\_dcsr, 126
- fasp\_format\_dcsr\_dbsr, 126
- fasp\_format\_dcsr\_dcoo, 127
- fasp\_format\_dcsr\_dcsr, 127
- fasp\_format\_dstr\_dbsr, 128
- fasp\_format\_dstr\_dcsr, 128
- BlaILU.c, 129
  - fasp\_iluk, 130
  - fasp\_ilut, 130
  - fasp\_ilutp, 131
  - fasp\_symbfactor, 133
- BlaILUSetupBSR.c, 135
  - fasp\_ilu\_dbsr\_setup, 136
  - fasp\_ilu\_dbsr\_setup\_levsch\_omp, 136
  - fasp\_ilu\_dbsr\_setup\_mc\_omp, 137
  - fasp\_ilu\_dbsr\_setup\_omp, 137
- BlaILUSetupCSR.c, 138
  - fasp\_ilu\_dcsr\_setup, 139
- BlaILUSetupSTR.c, 139
  - fasp\_ilu\_dstr\_setup0, 140
  - fasp\_ilu\_dstr\_setup1, 140
- BlaIO.c, 141
  - dlength, 164
  - fasp\_dbsr\_print, 143
  - fasp\_dbsr\_read, 143
  - fasp\_dbsr\_write, 144
  - fasp\_dbsr\_write\_coo, 144
  - fasp\_dcoo\_print, 145
  - fasp\_dcoo\_read, 145
  - fasp\_dcoo\_read1, 146
  - fasp\_dcoo\_shift\_read, 146
  - fasp\_dcoo\_write, 147
  - fasp\_dcsr\_print, 147
  - fasp\_dcsr\_read, 148
  - fasp\_dcsr\_write\_coo, 148
  - fasp\_dcsrvec\_read1, 149
  - fasp\_dcsrvec\_read2, 149
  - fasp\_dcsrvec\_write1, 151
  - fasp\_dcsrvec\_write2, 152
  - fasp\_dmtx\_read, 153
  - fasp\_dmtxsym\_read, 153
  - fasp\_dstr\_print, 154
  - fasp\_dstr\_read, 154
  - fasp\_dstr\_write, 155
  - fasp\_dvec\_print, 155
  - fasp\_dvec\_read, 156
  - fasp\_dvec\_write, 156
  - fasp\_dvecind\_write, 157
  - fasp\_hb\_read, 157
  - fasp\_ivec\_print, 158
  - fasp\_ivec\_read, 158
  - fasp\_ivec\_write, 159
  - fasp\_ivecind\_read, 159
  - fasp\_matrix\_read, 160
  - fasp\_matrix\_read\_bin, 161
  - fasp\_matrix\_write, 161
  - fasp\_vector\_read, 162
  - fasp\_vector\_write, 163
  - ilength, 164
- BlaOrderingCSR.c, 164
  - fasp\_dcsr\_CMK\_order, 164
  - fasp\_dcsr\_RCMK\_order, 165
- BlaSchwarzSetup.c, 165
  - fasp\_dcsr\_swz\_backward, 166
  - fasp\_dcsr\_swz\_forward, 166
  - fasp\_swz\_dcsr\_setup, 167
- BlaSmallMat.c, 167
  - fasp\_blas\_smat\_aAxpby, 169
  - fasp\_blas\_smat\_add, 170
  - fasp\_blas\_smat\_axm, 170
  - fasp\_blas\_smat\_mul, 171
  - fasp\_blas\_smat\_mul\_nc2, 171
  - fasp\_blas\_smat\_mul\_nc3, 172
  - fasp\_blas\_smat\_mul\_nc4, 172
  - fasp\_blas\_smat\_mul\_nc5, 173
  - fasp\_blas\_smat\_mul\_nc7, 173
  - fasp\_blas\_smat\_m xv, 174
  - fasp\_blas\_smat\_m xv\_nc2, 174
  - fasp\_blas\_smat\_m xv\_nc3, 175
  - fasp\_blas\_smat\_m xv\_nc4, 175
  - fasp\_blas\_smat\_m xv\_nc5, 176
  - fasp\_blas\_smat\_m xv\_nc7, 176
  - fasp\_blas\_smat\_ymAx, 176
  - fasp\_blas\_smat\_ymAx\_nc2, 177
  - fasp\_blas\_smat\_ymAx\_nc3, 177
  - fasp\_blas\_smat\_ymAx\_nc4, 178
  - fasp\_blas\_smat\_ymAx\_nc5, 178
  - fasp\_blas\_smat\_ymAx\_nc7, 179
  - fasp\_blas\_smat\_ypAx, 179
  - fasp\_blas\_smat\_ypAx\_nc2, 180
  - fasp\_blas\_smat\_ypAx\_nc3, 180
  - fasp\_blas\_smat\_ypAx\_nc4, 181
  - fasp\_blas\_smat\_ypAx\_nc5, 181
  - fasp\_blas\_smat\_ypAx\_nc7, 182
- BlaSmallMatInv.c, 182
  - fasp\_smat\_identity, 184
  - fasp\_smat\_identity\_nc2, 184
  - fasp\_smat\_identity\_nc3, 184
  - fasp\_smat\_identity\_nc5, 185
  - fasp\_smat\_identity\_nc7, 185
  - fasp\_smat\_inv, 185
  - fasp\_smat\_inv\_nc, 186
  - fasp\_smat\_inv\_nc2, 186
  - fasp\_smat\_inv\_nc3, 187
  - fasp\_smat\_inv\_nc4, 187
  - fasp\_smat\_inv\_nc5, 187

- fasp\_smat\_inv\_nc7, 188
  - fasp\_smat\_invp\_nc, 188
  - fasp\_smat\_Linf, 189
  - SWAP, 183
- BlaSmallMatLU.c, 189
  - fasp\_smat\_lu\_decomp, 190
  - fasp\_smat\_lu\_solve, 190
- BlaSparseBLC.c, 191
  - fasp\_dblc\_free, 192
- BlaSparseBSR.c, 192
  - fasp\_dbsr\_alloc, 193
  - fasp\_dbsr\_cp, 194
  - fasp\_dbsr\_create, 194
  - fasp\_dbsr\_diaginv, 195
  - fasp\_dbsr\_diaginv2, 195
  - fasp\_dbsr\_diaginv3, 196
  - fasp\_dbsr\_diaginv4, 196
  - fasp\_dbsr\_diagLU, 197
  - fasp\_dbsr\_diagLU2, 198
  - fasp\_dbsr\_diagpref, 198
  - fasp\_dbsr\_free, 199
  - fasp\_dbsr\_getblk, 199
  - fasp\_dbsr\_getdiag, 200
  - fasp\_dbsr\_getdiaginv, 200
  - fasp\_dbsr\_merge\_col, 202
  - fasp\_dbsr\_perm, 202
  - fasp\_dbsr\_trans, 203
- BlaSparseCheck.c, 203
  - fasp\_check\_dCSRmat, 204
  - fasp\_check\_diagdom, 204
  - fasp\_check\_diagpos, 205
  - fasp\_check\_diagzero, 205
  - fasp\_check\_iCSRmat, 206
  - fasp\_check\_ordering, 206
  - fasp\_check\_symm, 207
- BlaSparseCOO.c, 207
  - fasp\_dcoo\_alloc, 208
  - fasp\_dcoo\_create, 208
  - fasp\_dcoo\_free, 209
  - fasp\_dcoo\_shift, 209
- BlaSparseCSR.c, 210
  - fasp\_dcsr\_alloc, 211
  - fasp\_dcsr\_bandwidth, 212
  - fasp\_dcsr\_compress, 212
  - fasp\_dcsr\_compress\_inplace, 213
  - fasp\_dcsr\_cp, 213
  - fasp\_dcsr\_create, 214
  - fasp\_dcsr\_diagpref, 214
  - fasp\_dcsr\_free, 215
  - fasp\_dcsr\_getblk, 215
  - fasp\_dcsr\_getcol, 216
  - fasp\_dcsr\_getdiag, 216
  - fasp\_dcsr\_multicoloring, 218
  - fasp\_dcsr\_perm, 218
  - fasp\_dcsr\_permz, 219
  - fasp\_dcsr\_regdiag, 219
  - fasp\_dcsr\_shift, 221
  - fasp\_dcsr\_sort, 221
  - fasp\_dcsr\_sortz, 222
  - fasp\_dcsr\_symdiagscale, 222
  - fasp\_dcsr\_sympart, 223
  - fasp\_dcsr\_trans, 223
  - fasp\_dcsr\_transpose, 223
  - fasp\_dcsr\_transz, 224
  - fasp\_icsr\_cp, 225
  - fasp\_icsr\_create, 225
  - fasp\_icsr\_free, 226
  - fasp\_icsr\_trans, 226
- BlaSparseCSRL.c, 227
  - fasp\_dcsl\_create, 227
  - fasp\_dcsl\_free, 228
- BlaSparseSTR.c, 228
  - fasp\_dstr\_alloc, 229
  - fasp\_dstr\_cp, 229
  - fasp\_dstr\_create, 230
  - fasp\_dstr\_free, 231
- BlaSparseUtil.c, 231
  - fasp\_sparse\_aat\_, 232
  - fasp\_sparse\_abyb\_, 233
  - fasp\_sparse\_abybms\_, 233
  - fasp\_sparse\_aplbms\_, 234
  - fasp\_sparse\_aplusb\_, 235
  - fasp\_sparse\_iit\_, 235
  - fasp\_sparse\_mis, 236
  - fasp\_sparse\_rapcmp\_, 236
  - fasp\_sparse\_rapms\_, 237
  - fasp\_sparse\_wta\_, 238
  - fasp\_sparse\_wtams\_, 239
  - fasp\_sparse\_ytx\_, 239
  - fasp\_sparse\_ytxbig\_, 240
- BlaSpmvBLC.c, 240
  - fasp\_blas\_dblc\_aApy, 241
  - fasp\_blas\_dblc\_mxv, 241
- BlaSpmvBSR.c, 242
  - fasp\_blas\_dbsr\_aApyby, 243
  - fasp\_blas\_dbsr\_aApy, 243
  - fasp\_blas\_dbsr\_aApy\_agg, 245
  - fasp\_blas\_dbsr\_axm, 245
  - fasp\_blas\_dbsr\_mxm, 246
  - fasp\_blas\_dbsr\_mxv, 246
  - fasp\_blas\_dbsr\_mxv\_agg, 247
  - fasp\_blas\_dbsr\_rap, 247
  - fasp\_blas\_dbsr\_rap1, 248
  - fasp\_blas\_dbsr\_rap\_agg, 249
- BlaSpmvCSR.c, 249
  - fasp\_blas\_dcsr\_aApy, 251
  - fasp\_blas\_dcsr\_aApy\_agg, 251
  - fasp\_blas\_dcsr\_add, 252

- fasp\_blas\_dcsr\_axm, [252](#)
  - fasp\_blas\_dcsr\_mxm, [253](#)
  - fasp\_blas\_dcsr\_mxv, [253](#)
  - fasp\_blas\_dcsr\_mxv\_agg, [254](#)
  - fasp\_blas\_dcsr\_ptap, [254](#)
  - fasp\_blas\_dcsr\_rap, [255](#)
  - fasp\_blas\_dcsr\_rap2, [255](#)
  - fasp\_blas\_dcsr\_rap4, [256](#)
  - fasp\_blas\_dcsr\_rap\_agg, [257](#)
  - fasp\_blas\_dcsr\_rap\_agg1, [257](#)
  - fasp\_blas\_dcsr\_vmv, [258](#)
- BlaSpmvCSRL.c, [258](#)
  - fasp\_blas\_dcsr\_mxv, [259](#)
- BlaSpmvSTR.c, [259](#)
  - fasp\_blas\_dstr\_aAxy, [260](#)
  - fasp\_blas\_dstr\_diagscale, [260](#)
  - fasp\_blas\_dstr\_mxv, [261](#)
- BlaVector.c, [261](#)
  - fasp\_blas\_dvec\_axpy, [262](#)
  - fasp\_blas\_dvec\_axpyz, [262](#)
  - fasp\_blas\_dvec\_dotprod, [263](#)
  - fasp\_blas\_dvec\_norm1, [263](#)
  - fasp\_blas\_dvec\_norm2, [264](#)
  - fasp\_blas\_dvec\_norminf, [264](#)
  - fasp\_blas\_dvec\_reterr, [265](#)
- block\_dvector, [23](#)
  - fasp\_block.h, [274](#)
- block\_ivector, [23](#)
  - fasp\_block.h, [274](#)
- CF\_ORDER
  - fasp\_const.h, [278](#)
- CGPT
  - fasp\_const.h, [278](#)
- CLASSIC\_AMG
  - fasp\_const.h, [279](#)
- COARSE\_AC
  - fasp\_const.h, [279](#)
- COARSE\_CR
  - fasp\_const.h, [279](#)
- COARSE\_MIS
  - fasp\_const.h, [279](#)
- COARSE\_RS
  - fasp\_const.h, [279](#)
- COARSE\_RSP
  - fasp\_const.h, [279](#)
- CPFIRST
  - fasp\_const.h, [279](#)
- dBLMat, [24](#)
  - fasp\_block.h, [274](#)
- dBSRmat, [24](#)
  - fasp\_block.h, [274](#)
  - JA, [25](#)
  - val, [25](#)
- dCOOmat, [26](#)
  - fasp.h, [271](#)
- dCSRLmat, [26](#)
  - fasp.h, [272](#)
- dCSRmat, [27](#)
  - fasp.h, [272](#)
- dCSRmat2SAMGInput
  - XtrSamg.c, [474](#)
- ddenmat, [28](#)
  - fasp.h, [272](#)
- decoup\_type
  - input\_param, [47](#)
  - ITS\_param, [52](#)
- DESCEND
  - fasp\_const.h, [280](#)
- DIAGONAL\_PREF
  - fasp.h, [268](#)
- dlength
  - BlalO.c, [164](#)
- DLMALLOC
  - fasp.h, [268](#)
- doxygen.h, [265](#)
- dSTRmat, [29](#)
  - fasp.h, [272](#)
- dvector, [30](#)
  - fasp.h, [272](#)
- dvector2SAMGInput
  - XtrSamg.c, [474](#)
- e
  - grid2d, [31](#)
- edges
  - grid2d, [31](#)
- ediri
  - grid2d, [31](#)
- efather
  - grid2d, [31](#)
- ERROR\_ALLOC\_MEM
  - fasp\_const.h, [280](#)
- ERROR\_AMG\_COARSE\_TYPE
  - fasp\_const.h, [280](#)
- ERROR\_AMG\_COARSEING
  - fasp\_const.h, [280](#)
- ERROR\_AMG\_INTERP\_TYPE
  - fasp\_const.h, [280](#)
- ERROR\_AMG\_SETUP
  - fasp\_const.h, [280](#)
- ERROR\_AMG\_SMOOTH\_TYPE
  - fasp\_const.h, [280](#)
- ERROR\_DATA\_STRUCTURE
  - fasp\_const.h, [280](#)
- ERROR\_DATA\_ZERODIAG
  - fasp\_const.h, [281](#)
- ERROR\_DUMMY\_VAR

fasp\_const.h, [281](#)  
 ERROR\_INPUT\_PAR  
     fasp\_const.h, [281](#)  
 ERROR\_LIC\_TYPE  
     fasp\_const.h, [281](#)  
 ERROR\_MAT\_SIZE  
     fasp\_const.h, [281](#)  
 ERROR\_MISC  
     fasp\_const.h, [281](#)  
 ERROR\_NUM\_BLOCKS  
     fasp\_const.h, [281](#)  
 ERROR\_OPEN\_FILE  
     fasp\_const.h, [281](#)  
 ERROR\_QUAD\_DIM  
     fasp\_const.h, [282](#)  
 ERROR\_QUAD\_TYPE  
     fasp\_const.h, [282](#)  
 ERROR\_READ\_FILE  
     fasp\_const.h, [282](#)  
 ERROR\_REGRESS  
     fasp\_const.h, [282](#)  
 ERROR\_SOLVER\_EXIT  
     fasp\_const.h, [282](#)  
 ERROR\_SOLVER\_ILUSETUP  
     fasp\_const.h, [282](#)  
 ERROR\_SOLVER\_MAXIT  
     fasp\_const.h, [282](#)  
 ERROR\_SOLVER\_MISC  
     fasp\_const.h, [282](#)  
 ERROR\_SOLVER\_PRECTYPE  
     fasp\_const.h, [283](#)  
 ERROR\_SOLVER\_SOLSTAG  
     fasp\_const.h, [283](#)  
 ERROR\_SOLVER\_STAG  
     fasp\_const.h, [283](#)  
 ERROR\_SOLVER\_TOLSMALL  
     fasp\_const.h, [283](#)  
 ERROR\_SOLVER\_TYPE  
     fasp\_const.h, [283](#)  
 ERROR\_UNKNOWN  
     fasp\_const.h, [283](#)  
 ERROR\_WRONG\_FILE  
     fasp\_const.h, [283](#)  
  
 FALSE  
     fasp\_const.h, [283](#)  
 fasp.h, [265](#)  
     \_\_FASP\_HEADER\_\_, [268](#)  
     ABS, [268](#)  
     dCOOmat, [271](#)  
     dCSRLmat, [272](#)  
     dCSRmat, [272](#)  
     ddenmat, [272](#)  
     DIAGONAL\_PREF, [268](#)  
     DLMALLOC, [268](#)  
     dSTRmat, [272](#)  
     dvector, [272](#)  
     FASP\_VERSION, [269](#)  
     GE, [269](#)  
     GT, [269](#)  
     iCOOmat, [272](#)  
     iCSRmat, [272](#)  
     idenmat, [272](#)  
     INT, [269](#)  
     ISNAN, [269](#)  
     ivector, [272](#)  
     LE, [269](#)  
     LONG, [270](#)  
     LONGLONG, [270](#)  
     LS, [270](#)  
     MAX, [270](#)  
     MIN, [270](#)  
     NEDMALLOC, [270](#)  
     PUT\_INT, [270](#)  
     PUT\_REAL, [271](#)  
     REAL, [271](#)  
     RS\_C1, [271](#)  
     SHORT, [271](#)  
     STRLEN, [271](#)  
 fasp\_amg\_amli\_coef  
     PreMGRecurAMLI.c, [423](#)  
 fasp\_amg\_coarsening\_cr  
     PreAMGCoarsenCR.c, [380](#)  
 fasp\_amg\_coarsening\_rs  
     PreAMGCoarsenRS.c, [381](#)  
 fasp\_amg\_data\_bsr\_create  
     PreDataInit.c, [415](#)  
 fasp\_amg\_data\_bsr\_free  
     PreDataInit.c, [415](#)  
 fasp\_amg\_data\_create  
     PreDataInit.c, [416](#)  
 fasp\_amg\_data\_free  
     PreDataInit.c, [416](#)  
 fasp\_amg\_interp  
     PreAMGInterp.c, [382](#)  
 fasp\_amg\_interp\_em  
     PreAMGInterpEM.c, [383](#)  
 fasp\_amg\_setup\_cr  
     PreAMGSetupCR.c, [385](#)  
 fasp\_amg\_setup\_rs  
     PreAMGSetupRS.c, [386](#)  
 fasp\_amg\_setup\_sa  
     PreAMGSetupSA.c, [387](#)  
 fasp\_amg\_setup\_sa\_bsr  
     PreAMGSetupSABSR.c, [388](#)  
 fasp\_amg\_setup\_ua  
     PreAMGSetupUA.c, [389](#)  
 fasp\_amg\_setup\_ua\_bsr

PreAMGSetupUABSR.c, [390](#)

fasp\_amg\_solve  
  PreMGSolve.c, [426](#)

fasp\_amg\_solve\_amli  
  PreMGSolve.c, [426](#)

fasp\_amg\_solve\_namli  
  PreMGSolve.c, [427](#)

fasp\_amgcomplexity  
  AuxMessage.c, [84](#)

fasp\_amgcomplexity\_bsr  
  AuxMessage.c, [85](#)

fasp\_aux\_bbyteToldouble  
  AuxConvert.c, [74](#)

fasp\_aux\_BiSearch  
  AuxSort.c, [98](#)

fasp\_aux\_change\_endian4  
  AuxConvert.c, [74](#)

fasp\_aux\_change\_endian8  
  AuxConvert.c, [75](#)

fasp\_aux\_dQuickSort  
  AuxSort.c, [98](#)

fasp\_aux\_dQuickSortIndex  
  AuxSort.c, [100](#)

fasp\_aux\_givens  
  AuxGivens.c, [76](#)

fasp\_aux\_iQuickSort  
  AuxSort.c, [100](#)

fasp\_aux\_iQuickSortIndex  
  AuxSort.c, [101](#)

fasp\_aux\_merge  
  AuxSort.c, [102](#)

fasp\_aux\_msort  
  AuxSort.c, [102](#)

fasp\_aux\_unique  
  AuxSort.c, [103](#)

fasp\_blas\_darray\_ax  
  BlaArray.c, [115](#)

fasp\_blas\_darray\_axpby  
  BlaArray.c, [115](#)

fasp\_blas\_darray\_axpy  
  BlaArray.c, [116](#)

fasp\_blas\_darray\_axpy\_nc2  
  BlaArray.c, [116](#)

fasp\_blas\_darray\_axpy\_nc3  
  BlaArray.c, [117](#)

fasp\_blas\_darray\_axpy\_nc5  
  BlaArray.c, [117](#)

fasp\_blas\_darray\_axpy\_nc7  
  BlaArray.c, [118](#)

fasp\_blas\_darray\_axpyz  
  BlaArray.c, [118](#)

fasp\_blas\_darray\_axpyz\_nc2  
  BlaArray.c, [119](#)

fasp\_blas\_darray\_axpyz\_nc3  
  BlaArray.c, [119](#)

fasp\_blas\_darray\_axpyz\_nc5  
  BlaArray.c, [120](#)

fasp\_blas\_darray\_axpyz\_nc7  
  BlaArray.c, [120](#)

fasp\_blas\_darray\_dotprod  
  BlaArray.c, [121](#)

fasp\_blas\_darray\_norm1  
  BlaArray.c, [121](#)

fasp\_blas\_darray\_norm2  
  BlaArray.c, [122](#)

fasp\_blas\_darray\_norminf  
  BlaArray.c, [122](#)

fasp\_blas\_dblc\_aAxy  
  BlaSpmvBLC.c, [241](#)

fasp\_blas\_dblc\_mxv  
  BlaSpmvBLC.c, [241](#)

fasp\_blas\_dbsr\_aAxyby  
  BlaSpmvBSR.c, [243](#)

fasp\_blas\_dbsr\_aAxy  
  BlaSpmvBSR.c, [243](#)

fasp\_blas\_dbsr\_aAxy\_agg  
  BlaSpmvBSR.c, [245](#)

fasp\_blas\_dbsr\_axm  
  BlaSpmvBSR.c, [245](#)

fasp\_blas\_dbsr\_mxm  
  BlaSpmvBSR.c, [246](#)

fasp\_blas\_dbsr\_mxv  
  BlaSpmvBSR.c, [246](#)

fasp\_blas\_dbsr\_mxv\_agg  
  BlaSpmvBSR.c, [247](#)

fasp\_blas\_dbsr\_rap  
  BlaSpmvBSR.c, [247](#)

fasp\_blas\_dbsr\_rap1  
  BlaSpmvBSR.c, [248](#)

fasp\_blas\_dbsr\_rap\_agg  
  BlaSpmvBSR.c, [249](#)

fasp\_blas\_dcsr\_aAxy  
  BlaSpmvCSR.c, [251](#)

fasp\_blas\_dcsr\_aAxy\_agg  
  BlaSpmvCSR.c, [251](#)

fasp\_blas\_dcsr\_add  
  BlaSpmvCSR.c, [252](#)

fasp\_blas\_dcsr\_axm  
  BlaSpmvCSR.c, [252](#)

fasp\_blas\_dcsr\_mxm  
  BlaSpmvCSR.c, [253](#)

fasp\_blas\_dcsr\_mxv  
  BlaSpmvCSR.c, [253](#)

fasp\_blas\_dcsr\_mxv\_agg  
  BlaSpmvCSR.c, [254](#)

fasp\_blas\_dcsr\_ptap  
  BlaSpmvCSR.c, [254](#)

fasp\_blas\_dcsr\_rap

- BlaSpmvCSR.c, [255](#)
- fasp\_blas\_dcsr\_rap2
  - BlaSpmvCSR.c, [255](#)
- fasp\_blas\_dcsr\_rap4
  - BlaSpmvCSR.c, [256](#)
- fasp\_blas\_dcsr\_rap\_agg
  - BlaSpmvCSR.c, [257](#)
- fasp\_blas\_dcsr\_rap\_agg1
  - BlaSpmvCSR.c, [257](#)
- fasp\_blas\_dcsr\_vmv
  - BlaSpmvCSR.c, [258](#)
- fasp\_blas\_dcsr\_mxv
  - BlaSpmvCSR.c, [259](#)
- fasp\_blas\_dstr\_aAxy
  - BlaSpmvSTR.c, [260](#)
- fasp\_blas\_dstr\_diagscale
  - BlaSpmvSTR.c, [260](#)
- fasp\_blas\_dstr\_mxv
  - BlaSpmvSTR.c, [261](#)
- fasp\_blas\_dvec\_axpy
  - BlaVector.c, [262](#)
- fasp\_blas\_dvec\_axpyz
  - BlaVector.c, [262](#)
- fasp\_blas\_dvec\_dotprod
  - BlaVector.c, [263](#)
- fasp\_blas\_dvec\_norm1
  - BlaVector.c, [263](#)
- fasp\_blas\_dvec\_norm2
  - BlaVector.c, [264](#)
- fasp\_blas\_dvec\_norminf
  - BlaVector.c, [264](#)
- fasp\_blas\_dvec\_relerr
  - BlaVector.c, [265](#)
- fasp\_blas\_smat\_aAxyby
  - BlaSmallMat.c, [169](#)
- fasp\_blas\_smat\_add
  - BlaSmallMat.c, [170](#)
- fasp\_blas\_smat\_axm
  - BlaSmallMat.c, [170](#)
- fasp\_blas\_smat\_mul
  - BlaSmallMat.c, [171](#)
- fasp\_blas\_smat\_mul\_nc2
  - BlaSmallMat.c, [171](#)
- fasp\_blas\_smat\_mul\_nc3
  - BlaSmallMat.c, [172](#)
- fasp\_blas\_smat\_mul\_nc4
  - BlaSmallMat.c, [172](#)
- fasp\_blas\_smat\_mul\_nc5
  - BlaSmallMat.c, [173](#)
- fasp\_blas\_smat\_mul\_nc7
  - BlaSmallMat.c, [173](#)
- fasp\_blas\_smat\_mxv
  - BlaSmallMat.c, [174](#)
- fasp\_blas\_smat\_mxv\_nc2
  - BlaSmallMat.c, [174](#)
- fasp\_blas\_smat\_mxv\_nc3
  - BlaSmallMat.c, [175](#)
- fasp\_blas\_smat\_mxv\_nc4
  - BlaSmallMat.c, [175](#)
- fasp\_blas\_smat\_mxv\_nc5
  - BlaSmallMat.c, [176](#)
- fasp\_blas\_smat\_mxv\_nc7
  - BlaSmallMat.c, [176](#)
- fasp\_blas\_smat\_yMAx
  - BlaSmallMat.c, [176](#)
- fasp\_blas\_smat\_yMAx\_nc2
  - BlaSmallMat.c, [177](#)
- fasp\_blas\_smat\_yMAx\_nc3
  - BlaSmallMat.c, [177](#)
- fasp\_blas\_smat\_yMAx\_nc4
  - BlaSmallMat.c, [178](#)
- fasp\_blas\_smat\_yMAx\_nc5
  - BlaSmallMat.c, [178](#)
- fasp\_blas\_smat\_yMAx\_nc7
  - BlaSmallMat.c, [179](#)
- fasp\_blas\_smat\_yPAx
  - BlaSmallMat.c, [179](#)
- fasp\_blas\_smat\_yPAx\_nc2
  - BlaSmallMat.c, [180](#)
- fasp\_blas\_smat\_yPAx\_nc3
  - BlaSmallMat.c, [180](#)
- fasp\_blas\_smat\_yPAx\_nc4
  - BlaSmallMat.c, [181](#)
- fasp\_blas\_smat\_yPAx\_nc5
  - BlaSmallMat.c, [181](#)
- fasp\_blas\_smat\_yPAx\_nc7
  - BlaSmallMat.c, [182](#)
- fasp\_block.h, [272](#)
  - \_\_FASPBLOCK\_HEADER\_\_, [273](#)
  - block\_dvector, [274](#)
  - block\_ivecator, [274](#)
  - dBLCmat, [274](#)
  - dBSRmat, [274](#)
  - iBLCmat, [274](#)
- fasp\_check\_dCSRmat
  - BlaSparseCheck.c, [204](#)
- fasp\_check\_diagdom
  - BlaSparseCheck.c, [204](#)
- fasp\_check\_diagpos
  - BlaSparseCheck.c, [205](#)
- fasp\_check\_diagzero
  - BlaSparseCheck.c, [205](#)
- fasp\_check\_iCSRmat
  - BlaSparseCheck.c, [206](#)
- fasp\_check\_ordering
  - BlaSparseCheck.c, [206](#)
- fasp\_check\_symm
  - BlaSparseCheck.c, [207](#)



fasp\_chkerr  
  AuxMessage.c, [85](#)  
fasp\_const.h, [274](#)  
  AMLI\_CYCLE, [278](#)  
  ASCEND, [278](#)  
  BIGREAL, [278](#)  
  CF\_ORDER, [278](#)  
  CGPT, [278](#)  
  CLASSIC\_AMG, [279](#)  
  COARSE\_AC, [279](#)  
  COARSE\_CR, [279](#)  
  COARSE\_MIS, [279](#)  
  COARSE\_RS, [279](#)  
  COARSE\_RSP, [279](#)  
  CPFIRST, [279](#)  
  DESCEND, [280](#)  
  ERROR\_ALLOC\_MEM, [280](#)  
  ERROR\_AMG\_COARSE\_TYPE, [280](#)  
  ERROR\_AMG\_COARSEING, [280](#)  
  ERROR\_AMG\_INTERP\_TYPE, [280](#)  
  ERROR\_AMG\_SETUP, [280](#)  
  ERROR\_AMG\_SMOOTH\_TYPE, [280](#)  
  ERROR\_DATA\_STRUCTURE, [280](#)  
  ERROR\_DATA\_ZERODIAG, [281](#)  
  ERROR\_DUMMY\_VAR, [281](#)  
  ERROR\_INPUT\_PAR, [281](#)  
  ERROR\_LIC\_TYPE, [281](#)  
  ERROR\_MAT\_SIZE, [281](#)  
  ERROR\_MISC, [281](#)  
  ERROR\_NUM\_BLOCKS, [281](#)  
  ERROR\_OPEN\_FILE, [281](#)  
  ERROR\_QUAD\_DIM, [282](#)  
  ERROR\_QUAD\_TYPE, [282](#)  
  ERROR\_READ\_FILE, [282](#)  
  ERROR\_REGRESS, [282](#)  
  ERROR\_SOLVER\_EXIT, [282](#)  
  ERROR\_SOLVER\_ILUSETUP, [282](#)  
  ERROR\_SOLVER\_MAXIT, [282](#)  
  ERROR\_SOLVER\_MISC, [282](#)  
  ERROR\_SOLVER\_PRECTYPE, [283](#)  
  ERROR\_SOLVER\_SOLSTAG, [283](#)  
  ERROR\_SOLVER\_STAG, [283](#)  
  ERROR\_SOLVER\_TOLSMALL, [283](#)  
  ERROR\_SOLVER\_TYPE, [283](#)  
  ERROR\_UNKNOWN, [283](#)  
  ERROR\_WRONG\_FILE, [283](#)  
  FALSE, [283](#)  
  FASP\_SUCCESS, [284](#)  
  FGPT, [284](#)  
  FPFIRST, [284](#)  
  G0PT, [284](#)  
  ILUk, [284](#)  
  ILU<sub>t</sub>, [284](#)  
  ILU<sub>tp</sub>, [284](#)  
  INTERP\_DIR, [285](#)  
  INTERP\_ENG, [285](#)  
  INTERP\_EXT, [285](#)  
  INTERP\_STD, [285](#)  
  ISPT, [285](#)  
  MAT\_bBSR, [285](#)  
  MAT\_bCSR, [285](#)  
  MAT\_BLC, [285](#)  
  MAT\_BSR, [286](#)  
  MAT\_bSTR, [286](#)  
  MAT\_CSR, [286](#)  
  MAT\_CSRL, [286](#)  
  MAT\_FREE, [286](#)  
  MAT\_STR, [286](#)  
  MAT\_SymCSR, [286](#)  
  MAX\_AMG\_LVL, [286](#)  
  MAX\_CRATE, [287](#)  
  MAX\_REFINE\_LVL, [287](#)  
  MAX\_RESTART, [287](#)  
  MAX\_STAG, [287](#)  
  MIN\_CDOF, [287](#)  
  MIN\_CRATE, [287](#)  
  NL\_AMLI\_CYCLE, [287](#)  
  NO\_ORDER, [287](#)  
  NPAIR, [288](#)  
  OFF, [288](#)  
  ON, [288](#)  
  OPENMP\_HOLDS, [288](#)  
  PAIRWISE, [288](#)  
  PREC\_AMG, [288](#)  
  PREC\_DIAG, [288](#)  
  PREC\_FMG, [289](#)  
  PREC\_ILU, [289](#)  
  PREC\_NULL, [289](#)  
  PREC\_SCHWARZ, [289](#)  
  PRINT\_ALL, [289](#)  
  PRINT\_MIN, [289](#)  
  PRINT\_MORE, [289](#)  
  PRINT\_MOST, [289](#)  
  PRINT\_NONE, [290](#)  
  PRINT\_SOME, [290](#)  
  SA\_AMG, [290](#)  
  SCHWARZ\_BACKWARD, [290](#)  
  SCHWARZ\_FORWARD, [290](#)  
  SCHWARZ\_SYMMETRIC, [290](#)  
  SMALLREAL, [290](#)  
  SMALLREAL2, [291](#)  
  SMOOTHER\_BLKOil, [291](#)  
  SMOOTHER\_CG, [291](#)  
  SMOOTHER\_GS, [291](#)  
  SMOOTHER\_GSOR, [291](#)  
  SMOOTHER\_JACOBI, [291](#)  
  SMOOTHER\_L1DIAG, [291](#)  
  SMOOTHER\_POLY, [291](#)

SMOOTHER\_SGS, [292](#)  
 SMOOTHER\_SGSOR, [292](#)  
 SMOOTHER\_SOR, [292](#)  
 SMOOTHER\_SPETEN, [292](#)  
 SMOOTHER\_SSOR, [292](#)  
 SOLVER\_AMG, [292](#)  
 SOLVER\_BiCGstab, [292](#)  
 SOLVER\_CG, [292](#)  
 SOLVER\_DEFAULT, [293](#)  
 SOLVER\_FMG, [293](#)  
 SOLVER\_GCG, [293](#)  
 SOLVER\_GCR, [293](#)  
 SOLVER\_GMRES, [293](#)  
 SOLVER\_MinRes, [293](#)  
 SOLVER\_MUMPS, [293](#)  
 SOLVER\_PARDISO, [293](#)  
 SOLVER\_SBiCGstab, [294](#)  
 SOLVER\_SCG, [294](#)  
 SOLVER\_SGCG, [294](#)  
 SOLVER\_SGMRES, [294](#)  
 SOLVER\_SMinRes, [294](#)  
 SOLVER\_SUPERLU, [294](#)  
 SOLVER\_SVGMRES, [294](#)  
 SOLVER\_SVGMRES, [294](#)  
 SOLVER\_UMFPACK, [295](#)  
 SOLVER\_VFGMRES, [295](#)  
 SOLVER\_VGMRES, [295](#)  
 SPAIR, [295](#)  
 STAG\_RATIO, [295](#)  
 STOP\_MOD\_REL\_RES, [295](#)  
 STOP\_REL\_PRECRES, [295](#)  
 STOP\_REL\_RES, [295](#)  
 TRUE, [296](#)  
 UA\_AMG, [296](#)  
 UNPT, [296](#)  
 USERDEFINED, [296](#)  
 V\_CYCLE, [296](#)  
 VMB, [296](#)  
 VW\_CYCLE, [296](#)  
 W\_CYCLE, [297](#)  
 WV\_CYCLE, [297](#)  
 fasp\_cputime  
     AuxMessage.c, [85](#)  
 fasp\_darray\_cp  
     AuxArray.c, [72](#)  
 fasp\_darray\_set  
     AuxArray.c, [72](#)  
 fasp\_dblc\_free  
     BlaSparseBLC.c, [192](#)  
 fasp\_dbsr\_alloc  
     BlaSparseBSR.c, [193](#)  
 fasp\_dbsr\_cp  
     BlaSparseBSR.c, [194](#)  
 fasp\_dbsr\_create  
     BlaSparseBSR.c, [194](#)  
     fasp\_dbsr\_diaginv  
         BlaSparseBSR.c, [195](#)  
     fasp\_dbsr\_diaginv2  
         BlaSparseBSR.c, [195](#)  
     fasp\_dbsr\_diaginv3  
         BlaSparseBSR.c, [196](#)  
     fasp\_dbsr\_diaginv4  
         BlaSparseBSR.c, [196](#)  
     fasp\_dbsr\_diagLU  
         BlaSparseBSR.c, [197](#)  
     fasp\_dbsr\_diagLU2  
         BlaSparseBSR.c, [198](#)  
     fasp\_dbsr\_diagpref  
         BlaSparseBSR.c, [198](#)  
     fasp\_dbsr\_free  
         BlaSparseBSR.c, [199](#)  
     fasp\_dbsr\_getblk  
         BlaSparseBSR.c, [199](#)  
     fasp\_dbsr\_getdiag  
         BlaSparseBSR.c, [200](#)  
     fasp\_dbsr\_getdiaginv  
         BlaSparseBSR.c, [200](#)  
     fasp\_dbsr\_merge\_col  
         BlaSparseBSR.c, [202](#)  
     fasp\_dbsr\_perm  
         BlaSparseBSR.c, [202](#)  
     fasp\_dbsr\_plot  
         AuxGraphics.c, [77](#)  
     fasp\_dbsr\_print  
         BlalO.c, [143](#)  
     fasp\_dbsr\_read  
         BlalO.c, [143](#)  
     fasp\_dbsr\_subplot  
         AuxGraphics.c, [78](#)  
     fasp\_dbsr\_trans  
         BlaSparseBSR.c, [203](#)  
     fasp\_dbsr\_write  
         BlalO.c, [144](#)  
     fasp\_dbsr\_write\_coo  
         BlalO.c, [144](#)  
     fasp\_dcoo\_alloc  
         BlaSparseCOO.c, [208](#)  
     fasp\_dcoo\_create  
         BlaSparseCOO.c, [208](#)  
     fasp\_dcoo\_free  
         BlaSparseCOO.c, [209](#)  
     fasp\_dcoo\_print  
         BlalO.c, [145](#)  
     fasp\_dcoo\_read  
         BlalO.c, [145](#)  
     fasp\_dcoo\_read1  
         BlalO.c, [146](#)  
     fasp\_dcoo\_shift

BlaSparseCOO.c, 209  
fasp\_dcoo\_shift\_read  
  BlalO.c, 146  
fasp\_dcoo\_write  
  BlalO.c, 147  
fasp\_dcsr\_alloc  
  BlaSparseCSR.c, 211  
fasp\_dcsr\_bandwidth  
  BlaSparseCSR.c, 212  
fasp\_dcsr\_CMK\_order  
  BlaOrderingCSR.c, 164  
fasp\_dcsr\_compress  
  BlaSparseCSR.c, 212  
fasp\_dcsr\_compress\_inplace  
  BlaSparseCSR.c, 213  
fasp\_dcsr\_cp  
  BlaSparseCSR.c, 213  
fasp\_dcsr\_create  
  BlaSparseCSR.c, 214  
fasp\_dcsr\_diagpref  
  BlaSparseCSR.c, 214  
fasp\_dcsr\_free  
  BlaSparseCSR.c, 215  
fasp\_dcsr\_getblk  
  BlaSparseCSR.c, 215  
fasp\_dcsr\_getcol  
  BlaSparseCSR.c, 216  
fasp\_dcsr\_getdiag  
  BlaSparseCSR.c, 216  
fasp\_dcsr\_maxeig  
  BlaEigen.c, 123  
fasp\_dcsr\_multicoloring  
  BlaSparseCSR.c, 218  
fasp\_dcsr\_perm  
  BlaSparseCSR.c, 218  
fasp\_dcsr\_permz  
  BlaSparseCSR.c, 219  
fasp\_dcsr\_plot  
  AuxGraphics.c, 78  
fasp\_dcsr\_print  
  BlalO.c, 147  
fasp\_dcsr\_RCMK\_order  
  BlaOrderingCSR.c, 165  
fasp\_dcsr\_read  
  BlalO.c, 148  
fasp\_dcsr\_regdiag  
  BlaSparseCSR.c, 219  
fasp\_dcsr\_shift  
  BlaSparseCSR.c, 221  
fasp\_dcsr\_sort  
  BlaSparseCSR.c, 221  
fasp\_dcsr\_sortz  
  BlaSparseCSR.c, 222  
fasp\_dcsr\_subplot  
  AuxGraphics.c, 79  
fasp\_dcsr\_swz\_backward  
  BlaSchwarzSetup.c, 166  
fasp\_dcsr\_swz\_forward  
  BlaSchwarzSetup.c, 166  
fasp\_dcsr\_symdiagscale  
  BlaSparseCSR.c, 222  
fasp\_dcsr\_sympart  
  BlaSparseCSR.c, 223  
fasp\_dcsr\_trans  
  BlaSparseCSR.c, 223  
fasp\_dcsr\_transpose  
  BlaSparseCSR.c, 223  
fasp\_dcsr\_transz  
  BlaSparseCSR.c, 224  
fasp\_dcsr\_write\_coo  
  BlalO.c, 148  
fasp\_dcsr\_create  
  BlaSparseCSRL.c, 227  
fasp\_dcsr\_free  
  BlaSparseCSRL.c, 228  
fasp\_dcsrvec\_read1  
  BlalO.c, 149  
fasp\_dcsrvec\_read2  
  BlalO.c, 149  
fasp\_dcsrvec\_write1  
  BlalO.c, 151  
fasp\_dcsrvec\_write2  
  BlalO.c, 152  
fasp\_dmtx\_read  
  BlalO.c, 153  
fasp\_dmtxsym\_read  
  BlalO.c, 153  
fasp\_dstr\_alloc  
  BlaSparseSTR.c, 229  
fasp\_dstr\_cp  
  BlaSparseSTR.c, 229  
fasp\_dstr\_create  
  BlaSparseSTR.c, 230  
fasp\_dstr\_free  
  BlaSparseSTR.c, 231  
fasp\_dstr\_print  
  BlalO.c, 154  
fasp\_dstr\_read  
  BlalO.c, 154  
fasp\_dstr\_write  
  BlalO.c, 155  
fasp\_dvec\_alloc  
  AuxVector.c, 107  
fasp\_dvec\_cp  
  AuxVector.c, 108  
fasp\_dvec\_create  
  AuxVector.c, 108  
fasp\_dvec\_free

AuxVector.c, [108](#)  
 fasp\_dvec\_isnan  
     AuxVector.c, [109](#)  
 fasp\_dvec\_maxdiff  
     AuxVector.c, [109](#)  
 fasp\_dvec\_print  
     BlalO.c, [155](#)  
 fasp\_dvec\_rand  
     AuxVector.c, [110](#)  
 fasp\_dvec\_read  
     BlalO.c, [156](#)  
 fasp\_dvec\_set  
     AuxVector.c, [111](#)  
 fasp\_dvec\_symdiagscale  
     AuxVector.c, [111](#)  
 fasp\_dvec\_write  
     BlalO.c, [156](#)  
 fasp\_dvecind\_write  
     BlalO.c, [157](#)  
 fasp\_famg\_solve  
     PreMGSolve.c, [427](#)  
 fasp\_format\_dblc\_dcsr  
     BlaFormat.c, [125](#)  
 fasp\_format\_dbsr\_dcoo  
     BlaFormat.c, [125](#)  
 fasp\_format\_dbsr\_dcsr  
     BlaFormat.c, [125](#)  
 fasp\_format\_dcoo\_dcsr  
     BlaFormat.c, [126](#)  
 fasp\_format\_dcsr\_dbsr  
     BlaFormat.c, [126](#)  
 fasp\_format\_dcsr\_dcoo  
     BlaFormat.c, [127](#)  
 fasp\_format\_dcsr\_dcsr  
     BlaFormat.c, [127](#)  
 fasp\_format\_dstr\_dbsr  
     BlaFormat.c, [128](#)  
 fasp\_format\_dstr\_dcsr  
     BlaFormat.c, [128](#)  
 fasp\_fwrapper\_dcsr\_amg\_  
     SolWrapper.c, [467](#)  
 fasp\_fwrapper\_dcsr\_krylov\_amg\_  
     SolWrapper.c, [467](#)  
 fasp\_fwrapper\_dcsr\_krylov\_ilu\_  
     SolWrapper.c, [468](#)  
 fasp\_fwrapper\_dcsr\_pardiso\_  
     SolWrapper.c, [469](#)  
 fasp\_generate\_diaginv\_block  
     ltrSmootherSTR.c, [320](#)  
 fasp\_get\_start\_end  
     AuxThreads.c, [104](#)  
 fasp\_gettime  
     AuxTiming.c, [106](#)  
 fasp\_grid.h, [297](#)  
     \_\_FASPGRID\_HEADER\_\_, [297](#)  
     grid2d, [298](#)  
     pcgrid2d, [298](#)  
     pgrid2d, [298](#)  
 fasp\_grid2d\_plot  
     AuxGraphics.c, [79](#)  
 fasp\_hb\_read  
     BlalO.c, [157](#)  
 fasp\_iarray\_cp  
     AuxArray.c, [73](#)  
 fasp\_iarray\_set  
     AuxArray.c, [73](#)  
 fasp\_icsr\_cp  
     BlaSparseCSR.c, [225](#)  
 fasp\_icsr\_create  
     BlaSparseCSR.c, [225](#)  
 fasp\_icsr\_free  
     BlaSparseCSR.c, [226](#)  
 fasp\_icsr\_trans  
     BlaSparseCSR.c, [226](#)  
 fasp\_ilu\_data\_create  
     PreDataInit.c, [417](#)  
 fasp\_ilu\_data\_free  
     PreDataInit.c, [417](#)  
 fasp\_ilu\_dbsr\_setup  
     BlalLUSetupBSR.c, [136](#)  
 fasp\_ilu\_dbsr\_setup\_levsch\_omp  
     BlalLUSetupBSR.c, [136](#)  
 fasp\_ilu\_dbsr\_setup\_mc\_omp  
     BlalLUSetupBSR.c, [137](#)  
 fasp\_ilu\_dbsr\_setup\_omp  
     BlalLUSetupBSR.c, [137](#)  
 fasp\_ilu\_dcsr\_setup  
     BlalLUSetupCSR.c, [139](#)  
 fasp\_ilu\_dstr\_setup0  
     BlalLUSetupSTR.c, [140](#)  
 fasp\_ilu\_dstr\_setup1  
     BlalLUSetupSTR.c, [140](#)  
 fasp\_iluk  
     BlalLU.c, [130](#)  
 fasp\_ilut  
     BlalLU.c, [130](#)  
 fasp\_ilutp  
     BlalLU.c, [131](#)  
 fasp\_itinfo  
     AuxMessage.c, [86](#)  
 fasp\_ivec\_alloc  
     AuxVector.c, [112](#)  
 fasp\_ivec\_create  
     AuxVector.c, [112](#)  
 fasp\_ivec\_free  
     AuxVector.c, [112](#)  
 fasp\_ivec\_print  
     BlalO.c, [158](#)

`fasp_ivec_read`  
BlalO.c, [158](#)

`fasp_ivec_set`  
AuxVector.c, [113](#)

`fasp_ivec_write`  
BlalO.c, [159](#)

`fasp_ivecind_read`  
BlalO.c, [159](#)

`fasp_matrix_read`  
BlalO.c, [160](#)

`fasp_matrix_read_bin`  
BlalO.c, [161](#)

`fasp_matrix_write`  
BlalO.c, [161](#)

`fasp_mem_free`  
AuxMemory.c, [82](#)

`fasp_mem_iludata_check`  
AuxMemory.c, [82](#)

`fasp_mem_realloc`  
AuxMemory.c, [83](#)

`fasp_mem_usage`  
AuxMemory.c, [83](#)

`fasp_message`  
AuxMessage.c, [86](#)

`fasp_param_amg_init`  
AuxParam.c, [88](#)

`fasp_param_amg_print`  
AuxParam.c, [89](#)

`fasp_param_amg_set`  
AuxParam.c, [89](#)

`fasp_param_amg_to_prec`  
AuxParam.c, [89](#)

`fasp_param_amg_to_precbsr`  
AuxParam.c, [91](#)

`fasp_param_check`  
AuxInput.c, [80](#)

`fasp_param_ilu_init`  
AuxParam.c, [91](#)

`fasp_param_ilu_print`  
AuxParam.c, [92](#)

`fasp_param_ilu_set`  
AuxParam.c, [92](#)

`fasp_param_init`  
AuxParam.c, [92](#)

`fasp_param_input`  
AuxInput.c, [81](#)

`fasp_param_input_init`  
AuxParam.c, [93](#)

`fasp_param_prec_to_amg`  
AuxParam.c, [93](#)

`fasp_param_precbsr_to_amg`  
AuxParam.c, [94](#)

`fasp_param_set`  
AuxParam.c, [94](#)

`fasp_param_solver_init`  
AuxParam.c, [95](#)

`fasp_param_solver_print`  
AuxParam.c, [95](#)

`fasp_param_solver_set`  
AuxParam.c, [95](#)

`fasp_param_swz_init`  
AuxParam.c, [96](#)

`fasp_param_swz_print`  
AuxParam.c, [96](#)

`fasp_param_swz_set`  
AuxParam.c, [97](#)

`fasp_poisson_fgm1d`  
SolGMGPoisson.c, [452](#)

`fasp_poisson_fgm2d`  
SolGMGPoisson.c, [453](#)

`fasp_poisson_fgm3d`  
SolGMGPoisson.c, [454](#)

`fasp_poisson_gmg1d`  
SolGMGPoisson.c, [454](#)

`fasp_poisson_gmg2d`  
SolGMGPoisson.c, [456](#)

`fasp_poisson_gmg3d`  
SolGMGPoisson.c, [457](#)

`fasp_poisson_gmgcg1d`  
SolGMGPoisson.c, [457](#)

`fasp_poisson_gmgcg2d`  
SolGMGPoisson.c, [458](#)

`fasp_poisson_gmgcg3d`  
SolGMGPoisson.c, [459](#)

`fasp_precond_amg`  
PreCSR.c, [409](#)

`fasp_precond_amg_nk`  
PreCSR.c, [409](#)

`fasp_precond_amli`  
PreCSR.c, [410](#)

`fasp_precond_data_init`  
PreDataInit.c, [418](#)

`fasp_precond_dblc_diag_3`  
PreBLC.c, [392](#)

`fasp_precond_dblc_diag_3_amg`  
PreBLC.c, [392](#)

`fasp_precond_dblc_diag_4`  
PreBLC.c, [393](#)

`fasp_precond_dblc_lower_3`  
PreBLC.c, [393](#)

`fasp_precond_dblc_lower_3_amg`  
PreBLC.c, [395](#)

`fasp_precond_dblc_lower_4`  
PreBLC.c, [395](#)

`fasp_precond_dblc_SGS_3`  
PreBLC.c, [397](#)

`fasp_precond_dblc_SGS_3_amg`  
PreBLC.c, [397](#)

- fasp\_precond\_dblc\_sweeping  
PreBLC.c, [399](#)
- fasp\_precond\_dblc\_upper\_3  
PreBLC.c, [399](#)
- fasp\_precond\_dblc\_upper\_3\_amg  
PreBLC.c, [401](#)
- fasp\_precond\_dbsr\_amg  
PreBSR.c, [402](#)
- fasp\_precond\_dbsr\_amg\_nk  
PreBSR.c, [403](#)
- fasp\_precond\_dbsr\_diag  
PreBSR.c, [403](#)
- fasp\_precond\_dbsr\_diag\_nc2  
PreBSR.c, [404](#)
- fasp\_precond\_dbsr\_diag\_nc3  
PreBSR.c, [404](#)
- fasp\_precond\_dbsr\_diag\_nc5  
PreBSR.c, [405](#)
- fasp\_precond\_dbsr\_diag\_nc7  
PreBSR.c, [405](#)
- fasp\_precond\_dbsr\_ilu  
PreBSR.c, [406](#)
- fasp\_precond\_dbsr\_ilu\_ls\_omp  
PreBSR.c, [406](#)
- fasp\_precond\_dbsr\_ilu\_mc\_omp  
PreBSR.c, [407](#)
- fasp\_precond\_dbsr\_namli  
PreBSR.c, [407](#)
- fasp\_precond\_diag  
PreCSR.c, [410](#)
- fasp\_precond\_dstr\_blockgs  
PreSTR.c, [429](#)
- fasp\_precond\_dstr\_diag  
PreSTR.c, [429](#)
- fasp\_precond\_dstr\_ilu0  
PreSTR.c, [430](#)
- fasp\_precond\_dstr\_ilu0\_backward  
PreSTR.c, [430](#)
- fasp\_precond\_dstr\_ilu0\_forward  
PreSTR.c, [431](#)
- fasp\_precond\_dstr\_ilu1  
PreSTR.c, [431](#)
- fasp\_precond\_dstr\_ilu1\_backward  
PreSTR.c, [431](#)
- fasp\_precond\_dstr\_ilu1\_forward  
PreSTR.c, [432](#)
- fasp\_precond\_famg  
PreCSR.c, [411](#)
- fasp\_precond\_ilu  
PreCSR.c, [411](#)
- fasp\_precond\_ilu\_backward  
PreCSR.c, [412](#)
- fasp\_precond\_ilu\_forward  
PreCSR.c, [412](#)
- fasp\_precond\_namli  
PreCSR.c, [413](#)
- fasp\_precond\_setup  
PreCSR.c, [413](#)
- fasp\_precond\_swz  
PreCSR.c, [414](#)
- fasp\_set\_gs\_threads  
AuxThreads.c, [104](#)
- fasp\_smat\_identity  
BlaSmallMatInv.c, [184](#)
- fasp\_smat\_identity\_nc2  
BlaSmallMatInv.c, [184](#)
- fasp\_smat\_identity\_nc3  
BlaSmallMatInv.c, [184](#)
- fasp\_smat\_identity\_nc5  
BlaSmallMatInv.c, [185](#)
- fasp\_smat\_identity\_nc7  
BlaSmallMatInv.c, [185](#)
- fasp\_smat\_inv  
BlaSmallMatInv.c, [185](#)
- fasp\_smat\_inv\_nc  
BlaSmallMatInv.c, [186](#)
- fasp\_smat\_inv\_nc2  
BlaSmallMatInv.c, [186](#)
- fasp\_smat\_inv\_nc3  
BlaSmallMatInv.c, [187](#)
- fasp\_smat\_inv\_nc4  
BlaSmallMatInv.c, [187](#)
- fasp\_smat\_inv\_nc5  
BlaSmallMatInv.c, [187](#)
- fasp\_smat\_inv\_nc7  
BlaSmallMatInv.c, [188](#)
- fasp\_smat\_invp\_nc  
BlaSmallMatInv.c, [188](#)
- fasp\_smat\_Linf  
BlaSmallMatInv.c, [189](#)
- fasp\_smat\_lu\_decomp  
BlaSmallMatLU.c, [190](#)
- fasp\_smat\_lu\_solve  
BlaSmallMatLU.c, [190](#)
- fasp\_smoother\_dbsr\_gs  
ltrSmootherBSR.c, [299](#)
- fasp\_smoother\_dbsr\_gs1  
ltrSmootherBSR.c, [300](#)
- fasp\_smoother\_dbsr\_gs\_ascend  
ltrSmootherBSR.c, [300](#)
- fasp\_smoother\_dbsr\_gs\_ascend1  
ltrSmootherBSR.c, [301](#)
- fasp\_smoother\_dbsr\_gs\_descend  
ltrSmootherBSR.c, [301](#)
- fasp\_smoother\_dbsr\_gs\_descend1  
ltrSmootherBSR.c, [302](#)
- fasp\_smoother\_dbsr\_gs\_order1  
ltrSmootherBSR.c, [302](#)

`fasp_smoother_dbsr_gs_order2`  
    `ltrSmootherBSR.c`, [303](#)

`fasp_smoother_dbsr_ilu`  
    `ltrSmootherBSR.c`, [304](#)

`fasp_smoother_dbsr_jacobi`  
    `ltrSmootherBSR.c`, [304](#)

`fasp_smoother_dbsr_jacobi1`  
    `ltrSmootherBSR.c`, [305](#)

`fasp_smoother_dbsr_jacobi_setup`  
    `ltrSmootherBSR.c`, [305](#)

`fasp_smoother_dbsr_sor`  
    `ltrSmootherBSR.c`, [306](#)

`fasp_smoother_dbsr_sor1`  
    `ltrSmootherBSR.c`, [306](#)

`fasp_smoother_dbsr_sor_ascend`  
    `ltrSmootherBSR.c`, [307](#)

`fasp_smoother_dbsr_sor_descend`  
    `ltrSmootherBSR.c`, [307](#)

`fasp_smoother_dbsr_sor_order`  
    `ltrSmootherBSR.c`, [308](#)

`fasp_smoother_dcsr_gs`  
    `ltrSmootherCSR.c`, [310](#)

`fasp_smoother_dcsr_gs_cf`  
    `ltrSmootherCSR.c`, [311](#)

`fasp_smoother_dcsr_gscr`  
    `ltrSmootherCSRcr.c`, [316](#)

`fasp_smoother_dcsr_ilu`  
    `ltrSmootherCSR.c`, [311](#)

`fasp_smoother_dcsr_jacobi`  
    `ltrSmootherCSR.c`, [312](#)

`fasp_smoother_dcsr_kaczmarz`  
    `ltrSmootherCSR.c`, [312](#)

`fasp_smoother_dcsr_L1diag`  
    `ltrSmootherCSR.c`, [313](#)

`fasp_smoother_dcsr_poly`  
    `ltrSmootherCSRpoly.c`, [318](#)

`fasp_smoother_dcsr_poly_old`  
    `ltrSmootherCSRpoly.c`, [319](#)

`fasp_smoother_dcsr_sgs`  
    `ltrSmootherCSR.c`, [314](#)

`fasp_smoother_dcsr_sor`  
    `ltrSmootherCSR.c`, [314](#)

`fasp_smoother_dcsr_sor_cf`  
    `ltrSmootherCSR.c`, [315](#)

`fasp_smoother_dstr_gs`  
    `ltrSmootherSTR.c`, [321](#)

`fasp_smoother_dstr_gs1`  
    `ltrSmootherSTR.c`, [322](#)

`fasp_smoother_dstr_gs_ascend`  
    `ltrSmootherSTR.c`, [322](#)

`fasp_smoother_dstr_gs_cf`  
    `ltrSmootherSTR.c`, [323](#)

`fasp_smoother_dstr_gs_descend`  
    `ltrSmootherSTR.c`, [323](#)

`fasp_smoother_dstr_gs_order`  
    `ltrSmootherSTR.c`, [324](#)

`fasp_smoother_dstr_jacobi`  
    `ltrSmootherSTR.c`, [324](#)

`fasp_smoother_dstr_jacobi1`  
    `ltrSmootherSTR.c`, [325](#)

`fasp_smoother_dstr_sor`  
    `ltrSmootherSTR.c`, [325](#)

`fasp_smoother_dstr_sor1`  
    `ltrSmootherSTR.c`, [326](#)

`fasp_smoother_dstr_sor_ascend`  
    `ltrSmootherSTR.c`, [327](#)

`fasp_smoother_dstr_sor_cf`  
    `ltrSmootherSTR.c`, [327](#)

`fasp_smoother_dstr_sor_descend`  
    `ltrSmootherSTR.c`, [328](#)

`fasp_smoother_dstr_sor_order`  
    `ltrSmootherSTR.c`, [328](#)

`fasp_solver_amg`  
    `SolAMG.c`, [433](#)

`fasp_solver_amli`  
    `PreMGRecurAMLI.c`, [423](#)

`fasp_solver_dblc_itsolver`  
    `SolBLC.c`, [434](#)

`fasp_solver_dblc_krylov`  
    `SolBLC.c`, [435](#)

`fasp_solver_dblc_krylov_block3`  
    `SolBLC.c`, [436](#)

`fasp_solver_dblc_krylov_block4`  
    `SolBLC.c`, [436](#)

`fasp_solver_dblc_krylov_sweeping`  
    `SolBLC.c`, [437](#)

`fasp_solver_dblc_pbcgs`  
    `KryPbcgs.c`, [330](#)

`fasp_solver_dblc_pcg`  
    `KryPcg.c`, [335](#)

`fasp_solver_dblc_pgcr`  
    `KryPgcr.c`, [342](#)

`fasp_solver_dblc_pgmres`  
    `KryPgmres.c`, [344](#)

`fasp_solver_dblc_pminres`  
    `KryPminres.c`, [349](#)

`fasp_solver_dblc_pvfgmres`  
    `KryPvfgmres.c`, [353](#)

`fasp_solver_dblc_pvgmres`  
    `KryPvgmres.c`, [357](#)

`fasp_solver_dblc_spbcg`  
    `KrySPbcgs.c`, [362](#)

`fasp_solver_dblc_spcg`  
    `KrySPcg.c`, [366](#)

`fasp_solver_dblc_spgmres`  
    `KrySPgmres.c`, [369](#)

`fasp_solver_dblc_spmminres`  
    `KrySPminres.c`, [373](#)

fasp\_solver\_dblc\_spgvmres  
KrySPvgmres.c, [376](#)

fasp\_solver\_dbsr\_itsolver  
SolBSR.c, [439](#)

fasp\_solver\_dbsr\_krylov  
SolBSR.c, [439](#)

fasp\_solver\_dbsr\_krylov\_amg  
SolBSR.c, [440](#)

fasp\_solver\_dbsr\_krylov\_amg\_nk  
SolBSR.c, [441](#)

fasp\_solver\_dbsr\_krylov\_diag  
SolBSR.c, [441](#)

fasp\_solver\_dbsr\_krylov\_ilu  
SolBSR.c, [442](#)

fasp\_solver\_dbsr\_krylov\_nk\_amg  
SolBSR.c, [442](#)

fasp\_solver\_dbsr\_pbcgs  
KryPbcgs.c, [331](#)

fasp\_solver\_dbsr\_pcg  
KryPcg.c, [336](#)

fasp\_solver\_dbsr\_pgmres  
KryPgmres.c, [345](#)

fasp\_solver\_dbsr\_pvfgmres  
KryPvfgmres.c, [354](#)

fasp\_solver\_dbsr\_pvgmres  
KryPvgmres.c, [358](#)

fasp\_solver\_dbsr\_spgmres  
KrySPbcgs.c, [363](#)

fasp\_solver\_dbsr\_spgmres  
KrySPgmres.c, [370](#)

fasp\_solver\_dbsr\_spgvmres  
KrySPvgmres.c, [377](#)

fasp\_solver\_dcsr\_itsolver  
SolCSR.c, [444](#)

fasp\_solver\_dcsr\_itsolver\_s  
SolCSR.c, [445](#)

fasp\_solver\_dcsr\_krylov  
SolCSR.c, [446](#)

fasp\_solver\_dcsr\_krylov\_amg  
SolCSR.c, [446](#)

fasp\_solver\_dcsr\_krylov\_amg\_nk  
SolCSR.c, [447](#)

fasp\_solver\_dcsr\_krylov\_diag  
SolCSR.c, [447](#)

fasp\_solver\_dcsr\_krylov\_ilu  
SolCSR.c, [448](#)

fasp\_solver\_dcsr\_krylov\_ilu\_M  
SolCSR.c, [449](#)

fasp\_solver\_dcsr\_krylov\_s  
SolCSR.c, [449](#)

fasp\_solver\_dcsr\_krylov\_swz  
SolCSR.c, [450](#)

fasp\_solver\_dcsr\_pbcgs  
KryPbcgs.c, [331](#)

fasp\_solver\_dcsr\_pcg  
KryPcg.c, [336](#)

fasp\_solver\_dcsr\_pgcg  
KryPgcg.c, [339](#)

fasp\_solver\_dcsr\_pgcr  
KryPgcr.c, [342](#)

fasp\_solver\_dcsr\_pgmres  
KryPgmres.c, [346](#)

fasp\_solver\_dcsr\_pminres  
KryPminres.c, [350](#)

fasp\_solver\_dcsr\_pvfgmres  
KryPvfgmres.c, [355](#)

fasp\_solver\_dcsr\_pvgmres  
KryPvgmres.c, [359](#)

fasp\_solver\_dcsr\_spgmres  
KrySPbcgs.c, [364](#)

fasp\_solver\_dcsr\_spcg  
KrySPcg.c, [367](#)

fasp\_solver\_dcsr\_spgmres  
KrySPgmres.c, [370](#)

fasp\_solver\_dcsr\_spmminres  
KrySPminres.c, [373](#)

fasp\_solver\_dcsr\_spgvmres  
KrySPvgmres.c, [377](#)

fasp\_solver\_dstr\_itsolver  
SolSTR.c, [463](#)

fasp\_solver\_dstr\_krylov  
SolSTR.c, [463](#)

fasp\_solver\_dstr\_krylov\_blockgs  
SolSTR.c, [464](#)

fasp\_solver\_dstr\_krylov\_diag  
SolSTR.c, [465](#)

fasp\_solver\_dstr\_krylov\_ilu  
SolSTR.c, [465](#)

fasp\_solver\_dstr\_pbcgs  
KryPbcgs.c, [332](#)

fasp\_solver\_dstr\_pcg  
KryPcg.c, [337](#)

fasp\_solver\_dstr\_pgmres  
KryPgmres.c, [347](#)

fasp\_solver\_dstr\_pminres  
KryPminres.c, [350](#)

fasp\_solver\_dstr\_pvgmres  
KryPvgmres.c, [360](#)

fasp\_solver\_dstr\_spgmres  
KrySPbcgs.c, [364](#)

fasp\_solver\_dstr\_spcg  
KrySPcg.c, [367](#)

fasp\_solver\_dstr\_spgmres  
KrySPgmres.c, [371](#)

fasp\_solver\_dstr\_spmminres  
KrySPminres.c, [374](#)

fasp\_solver\_dstr\_spgvmres  
KrySPvgmres.c, [378](#)



fasp\_solver\_famg  
    SolFAMG.c, [451](#)  
fasp\_solver\_fmecycle  
    PreMGCycleFull.c, [420](#)  
fasp\_solver\_itsolver  
    SolMatFree.c, [460](#)  
fasp\_solver\_krylov  
    SolMatFree.c, [461](#)  
fasp\_solver\_matfree\_init  
    SolMatFree.c, [461](#)  
fasp\_solver\_mgcycle  
    PreMGCycle.c, [419](#)  
fasp\_solver\_mgcycle\_bsr  
    PreMGCycle.c, [419](#)  
fasp\_solver\_mgrecur  
    PreMGRrecur.c, [422](#)  
fasp\_solver\_mumps  
    XtrMumps.c, [470](#)  
fasp\_solver\_mumps\_steps  
    XtrMumps.c, [472](#)  
fasp\_solver\_namli  
    PreMGRrecurAMLI.c, [424](#)  
fasp\_solver\_namli\_bsr  
    PreMGRrecurAMLI.c, [424](#)  
fasp\_solver\_pardiso  
    XtrPardiso.c, [473](#)  
fasp\_solver\_pbcgs  
    KryPbcgs.c, [333](#)  
fasp\_solver\_pcg  
    KryPcg.c, [338](#)  
fasp\_solver\_pgcg  
    KryPgcg.c, [340](#)  
fasp\_solver\_pgmres  
    KryPgmres.c, [347](#)  
fasp\_solver\_pminres  
    KryPminres.c, [351](#)  
fasp\_solver\_pvfgmres  
    KryPvfgmres.c, [355](#)  
fasp\_solver\_pvgmres  
    KryPvgmres.c, [360](#)  
fasp\_solver\_superlu  
    XtrSuperlu.c, [475](#)  
fasp\_solver\_umfpack  
    XtrUmfpack.c, [476](#)  
fasp\_sparse\_aat\_  
    BlaSparseUtil.c, [232](#)  
fasp\_sparse\_abyb\_  
    BlaSparseUtil.c, [233](#)  
fasp\_sparse\_abybms\_  
    BlaSparseUtil.c, [233](#)  
fasp\_sparse\_aplbms\_  
    BlaSparseUtil.c, [234](#)  
fasp\_sparse\_aplusb\_  
    BlaSparseUtil.c, [235](#)  
fasp\_sparse\_iit\_  
    BlaSparseUtil.c, [235](#)  
fasp\_sparse\_mis  
    BlaSparseUtil.c, [236](#)  
fasp\_sparse\_rapcmp\_  
    BlaSparseUtil.c, [236](#)  
fasp\_sparse\_rapms\_  
    BlaSparseUtil.c, [237](#)  
fasp\_sparse\_wta\_  
    BlaSparseUtil.c, [238](#)  
fasp\_sparse\_wtams\_  
    BlaSparseUtil.c, [239](#)  
fasp\_sparse\_ytx\_  
    BlaSparseUtil.c, [239](#)  
fasp\_sparse\_ytxbig\_  
    BlaSparseUtil.c, [240](#)  
FASP\_SUCCESS  
    fasp\_const.h, [284](#)  
fasp\_swz\_data\_free  
    PreDataInit.c, [418](#)  
fasp\_swz\_dcsr\_setup  
    BlaSchwarzSetup.c, [167](#)  
fasp\_symbfactor  
    BlalLU.c, [133](#)  
fasp\_vector\_read  
    BlalO.c, [162](#)  
fasp\_vector\_write  
    BlalO.c, [163](#)  
FASP\_VERSION  
    fasp.h, [269](#)  
FGPT  
    fasp\_const.h, [284](#)  
FPFIRST  
    fasp\_const.h, [284](#)  
G0PT  
    fasp\_const.h, [284](#)  
GE  
    fasp.h, [269](#)  
grid2d, [30](#)  
    e, [31](#)  
    edges, [31](#)  
    ediri, [31](#)  
    efather, [31](#)  
    fasp\_grid.h, [298](#)  
    p, [32](#)  
    pdiri, [32](#)  
    pfather, [32](#)  
    s, [32](#)  
    t, [32](#)  
    tfather, [33](#)  
    triangles, [33](#)  
    vertices, [33](#)  
GT

- fasp.h, 269
- iBLCmat, 33
  - fasp\_block.h, 274
- ICNTL
  - XtrMumps.c, 470
- iCOOmat, 34
  - fasp.h, 272
- iCSRmat, 35
  - fasp.h, 272
- idenmat, 36
  - fasp.h, 272
- ilength
  - BlalO.c, 164
- ILU\_data, 36
- ILU\_droptol
  - input\_param, 47
- ILU\_lfil
  - input\_param, 47
- ILU\_param, 38
- ILU\_permtol
  - input\_param, 47
- ILU\_relax
  - input\_param, 47
- ilu\_solve\_time
  - ltrSmootherBSR.c, 309
- ILU\_type
  - input\_param, 48
- ILUk
  - fasp\_const.h, 284
- ILUt
  - fasp\_const.h, 284
- ILUtp
  - fasp\_const.h, 284
- inifile
  - input\_param, 48
- input\_param, 39
  - AMG\_aggregation\_type, 40
  - AMG\_aggressive\_level, 40
  - AMG\_aggressive\_path, 40
  - AMG\_amli\_degree, 41
  - AMG\_coarse\_dof, 41
  - AMG\_coarse\_scaling, 41
  - AMG\_coarse\_solver, 41
  - AMG\_coarsening\_type, 41
  - AMG\_cycle\_type, 42
  - AMG\_ILU\_levels, 42
  - AMG\_interpolation\_type, 42
  - AMG\_levels, 42
  - AMG\_max\_aggregation, 42
  - AMG\_max\_row\_sum, 43
  - AMG\_maxit, 43
  - AMG\_nl\_amli\_krylov\_type, 43
  - AMG\_pair\_number, 43
  - AMG\_polynomial\_degree, 43
  - AMG\_postsmooth\_iter, 44
  - AMG\_presmooth\_iter, 44
  - AMG\_quality\_bound, 44
  - AMG\_relaxation, 44
  - AMG\_smooth\_filter, 44
  - AMG\_smooth\_order, 45
  - AMG\_smooth\_restriction, 45
  - AMG\_smoother, 45
  - AMG\_strong\_coupled, 45
  - AMG\_strong\_threshold, 45
  - AMG\_SWZ\_levels, 46
  - AMG\_tentative\_smooth, 46
  - AMG\_tol, 46
  - AMG\_truncation\_threshold, 46
  - AMG\_type, 46
  - decoup\_type, 47
  - ILU\_droptol, 47
  - ILU\_lfil, 47
  - ILU\_permtol, 47
  - ILU\_relax, 47
  - ILU\_type, 48
  - inifile, 48
  - itsolver\_maxit, 48
  - itsolver\_tol, 48
  - output\_type, 48
  - precond\_type, 49
  - print\_level, 49
  - problem\_num, 49
  - restart, 49
  - solver\_type, 49
  - stop\_type, 50
  - SWZ\_blksolver, 50
  - SWZ\_maxlvl, 50
  - SWZ\_mmsize, 50
  - SWZ\_type, 50
  - workdir, 51
- INT
  - fasp.h, 269
- INTERP\_DIR
  - fasp\_const.h, 285
- INTERP\_ENG
  - fasp\_const.h, 285
- INTERP\_EXT
  - fasp\_const.h, 285
- INTERP\_STD
  - fasp\_const.h, 285
- ISNAN
  - fasp.h, 269
- ISPT
  - fasp\_const.h, 285
- ltrSmootherBSR.c, 298
  - fasp\_smoother\_dbsr\_gs, 299
  - fasp\_smoother\_dbsr\_gs1, 300

- fasp\_smoother\_dbsr\_gs\_ascend, [300](#)
  - fasp\_smoother\_dbsr\_gs\_ascend1, [301](#)
  - fasp\_smoother\_dbsr\_gs\_descend, [301](#)
  - fasp\_smoother\_dbsr\_gs\_descend1, [302](#)
  - fasp\_smoother\_dbsr\_gs\_order1, [302](#)
  - fasp\_smoother\_dbsr\_gs\_order2, [303](#)
  - fasp\_smoother\_dbsr\_ilu, [304](#)
  - fasp\_smoother\_dbsr\_jacobi, [304](#)
  - fasp\_smoother\_dbsr\_jacobi1, [305](#)
  - fasp\_smoother\_dbsr\_jacobi\_setup, [305](#)
  - fasp\_smoother\_dbsr\_sor, [306](#)
  - fasp\_smoother\_dbsr\_sor1, [306](#)
  - fasp\_smoother\_dbsr\_sor\_ascend, [307](#)
  - fasp\_smoother\_dbsr\_sor\_descend, [307](#)
  - fasp\_smoother\_dbsr\_sor\_order, [308](#)
  - ilu\_solve\_time, [309](#)
- ltrSmootherCSR.c, [309](#)
  - fasp\_smoother\_dcsr\_gs, [310](#)
  - fasp\_smoother\_dcsr\_gs\_cf, [311](#)
  - fasp\_smoother\_dcsr\_ilu, [311](#)
  - fasp\_smoother\_dcsr\_jacobi, [312](#)
  - fasp\_smoother\_dcsr\_kaczmarz, [312](#)
  - fasp\_smoother\_dcsr\_L1diag, [313](#)
  - fasp\_smoother\_dcsr\_sgs, [314](#)
  - fasp\_smoother\_dcsr\_sor, [314](#)
  - fasp\_smoother\_dcsr\_sor\_cf, [315](#)
- ltrSmootherCSRcr.c, [316](#)
  - fasp\_smoother\_dcsr\_gscr, [316](#)
- ltrSmootherCSRpoly.c, [317](#)
  - fasp\_smoother\_dcsr\_poly, [318](#)
  - fasp\_smoother\_dcsr\_poly\_old, [319](#)
- ltrSmootherSTR.c, [319](#)
  - fasp\_generate\_diaginv\_block, [320](#)
  - fasp\_smoother\_dstr\_gs, [321](#)
  - fasp\_smoother\_dstr\_gs1, [322](#)
  - fasp\_smoother\_dstr\_gs\_ascend, [322](#)
  - fasp\_smoother\_dstr\_gs\_cf, [323](#)
  - fasp\_smoother\_dstr\_gs\_descend, [323](#)
  - fasp\_smoother\_dstr\_gs\_order, [324](#)
  - fasp\_smoother\_dstr\_jacobi, [324](#)
  - fasp\_smoother\_dstr\_jacobi1, [325](#)
  - fasp\_smoother\_dstr\_sor, [325](#)
  - fasp\_smoother\_dstr\_sor1, [326](#)
  - fasp\_smoother\_dstr\_sor\_ascend, [327](#)
  - fasp\_smoother\_dstr\_sor\_cf, [327](#)
  - fasp\_smoother\_dstr\_sor\_descend, [328](#)
  - fasp\_smoother\_dstr\_sor\_order, [328](#)
- ITS\_param, [51](#)
  - decoup\_type, [52](#)
  - itsolver\_type, [52](#)
  - maxit, [52](#)
  - precond\_type, [52](#)
  - print\_level, [52](#)
  - restart, [53](#)
  - stop\_type, [53](#)
  - tol, [53](#)
- itsolver\_maxit
  - input\_param, [48](#)
- itsolver\_tol
  - input\_param, [48](#)
- itsolver\_type
  - ITS\_param, [52](#)
- ivector, [53](#)
  - fasp.h, [272](#)
- JA
  - dBSRmat, [25](#)
- KryPbcgs.c, [329](#)
  - fasp\_solver\_dblc\_pbcgs, [330](#)
  - fasp\_solver\_dbsr\_pbcgs, [331](#)
  - fasp\_solver\_dcsr\_pbcgs, [331](#)
  - fasp\_solver\_dstr\_pbcgs, [332](#)
  - fasp\_solver\_pbcgs, [333](#)
- KryPcg.c, [334](#)
  - fasp\_solver\_dblc\_pcg, [335](#)
  - fasp\_solver\_dbsr\_pcg, [336](#)
  - fasp\_solver\_dcsr\_pcg, [336](#)
  - fasp\_solver\_dstr\_pcg, [337](#)
  - fasp\_solver\_pcg, [338](#)
- KryPgcr.c, [339](#)
  - fasp\_solver\_dcsr\_pgcr, [339](#)
  - fasp\_solver\_pgcr, [340](#)
- KryPgcr.c, [341](#)
  - fasp\_solver\_dblc\_pgcr, [342](#)
  - fasp\_solver\_dcsr\_pgcr, [342](#)
- KryPgmres.c, [343](#)
  - fasp\_solver\_dblc\_pgmres, [344](#)
  - fasp\_solver\_dbsr\_pgmres, [345](#)
  - fasp\_solver\_dcsr\_pgmres, [346](#)
  - fasp\_solver\_dstr\_pgmres, [347](#)
  - fasp\_solver\_pgmres, [347](#)
- KryPminres.c, [348](#)
  - fasp\_solver\_dblc\_pminres, [349](#)
  - fasp\_solver\_dcsr\_pminres, [350](#)
  - fasp\_solver\_dstr\_pminres, [350](#)
  - fasp\_solver\_pminres, [351](#)
- KryPvfgmres.c, [352](#)
  - fasp\_solver\_dblc\_pvfgmres, [353](#)
  - fasp\_solver\_dbsr\_pvfgmres, [354](#)
  - fasp\_solver\_dcsr\_pvfgmres, [355](#)
  - fasp\_solver\_pvfgmres, [355](#)
- KryPvgmres.c, [356](#)
  - fasp\_solver\_dblc\_pvgmres, [357](#)
  - fasp\_solver\_dbsr\_pvgmres, [358](#)
  - fasp\_solver\_dcsr\_pvgmres, [359](#)
  - fasp\_solver\_dstr\_pvgmres, [360](#)
  - fasp\_solver\_pvgmres, [360](#)
- KrySPbcgs.c, [361](#)

- fasp\_solver\_dblc\_spgbcs, [362](#)
  - fasp\_solver\_dbsr\_spgbcs, [363](#)
  - fasp\_solver\_dcsr\_spgbcs, [364](#)
  - fasp\_solver\_dstr\_spgbcs, [364](#)
- KrySPcg.c, [365](#)
  - fasp\_solver\_dblc\_spcg, [366](#)
  - fasp\_solver\_dcsr\_spcg, [367](#)
  - fasp\_solver\_dstr\_spcg, [367](#)
- KrySPgmres.c, [368](#)
  - fasp\_solver\_dblc\_spgmres, [369](#)
  - fasp\_solver\_dbsr\_spgmres, [370](#)
  - fasp\_solver\_dcsr\_spgmres, [370](#)
  - fasp\_solver\_dstr\_spgmres, [371](#)
- KrySPminres.c, [372](#)
  - fasp\_solver\_dblc\_spmminres, [373](#)
  - fasp\_solver\_dcsr\_spmminres, [373](#)
  - fasp\_solver\_dstr\_spmminres, [374](#)
- KrySPvgmres.c, [375](#)
  - fasp\_solver\_dblc\_spvvgmres, [376](#)
  - fasp\_solver\_dbsr\_spvvgmres, [377](#)
  - fasp\_solver\_dcsr\_spvvgmres, [377](#)
  - fasp\_solver\_dstr\_spvvgmres, [378](#)
- LE
  - fasp.h, [269](#)
- local\_A
  - precond\_data\_sweeping, [65](#)
- local\_index
  - precond\_data\_sweeping, [65](#)
- local\_LU
  - precond\_data\_sweeping, [65](#)
- LONG
  - fasp.h, [270](#)
- LONGLONG
  - fasp.h, [270](#)
- LS
  - fasp.h, [270](#)
- LU\_diag
  - precond\_data\_blc, [59](#)
- MAT\_bBSR
  - fasp\_const.h, [285](#)
- MAT\_bCSR
  - fasp\_const.h, [285](#)
- MAT\_BLC
  - fasp\_const.h, [285](#)
- MAT\_BSR
  - fasp\_const.h, [286](#)
- MAT\_bSTR
  - fasp\_const.h, [286](#)
- MAT\_CSR
  - fasp\_const.h, [286](#)
- MAT\_CSRL
  - fasp\_const.h, [286](#)
- MAT\_FREE
  - fasp\_const.h, [286](#)
- MAT\_STR
  - fasp\_const.h, [286](#)
- MAT\_SymCSR
  - fasp\_const.h, [286](#)
- MAX
  - fasp.h, [270](#)
- MAX\_AMG\_LVL
  - fasp\_const.h, [286](#)
- MAX\_CRATE
  - fasp\_const.h, [287](#)
- MAX\_REFINE\_LVL
  - fasp\_const.h, [287](#)
- MAX\_RESTART
  - fasp\_const.h, [287](#)
- MAX\_STAG
  - fasp\_const.h, [287](#)
- maxit
  - ITS\_param, [52](#)
- mgl
  - precond\_data\_blc, [59](#)
- MIN
  - fasp.h, [270](#)
- MIN\_CDOF
  - fasp\_const.h, [287](#)
- MIN\_CRATE
  - fasp\_const.h, [287](#)
- Mumps\_data, [54](#)
- mxv\_matfree, [55](#)
- NEDMALLOC
  - fasp.h, [270](#)
- NL\_AMLI\_CYCLE
  - fasp\_const.h, [287](#)
- NO\_ORDER
  - fasp\_const.h, [287](#)
- NPAIR
  - fasp\_const.h, [288](#)
- NumLayers
  - precond\_data\_sweeping, [65](#)
- OFF
  - fasp\_const.h, [288](#)
- ON
  - fasp\_const.h, [288](#)
- OPENMP\_HOLDS
  - fasp\_const.h, [288](#)
- output\_type
  - input\_param, [48](#)
- p
  - grid2d, [32](#)
- PAIRWISE
  - fasp\_const.h, [288](#)
- Pardiso\_data, [55](#)

- pcgrid2d
  - fasp\_grid.h, 298
- pdiri
  - grid2d, 32
- pfather
  - grid2d, 32
- pgrid2d
  - fasp\_grid.h, 298
- PreAMGCoarsenCR.c, 379
  - fasp\_amg\_coarsening\_cr, 380
- PreAMGCoarsenRS.c, 380
  - fasp\_amg\_coarsening\_rs, 381
- PreAMGInterp.c, 382
  - fasp\_amg\_interp, 382
- PreAMGInterpEM.c, 383
  - fasp\_amg\_interp\_em, 383
- PreAMGSetupCR.c, 384
  - fasp\_amg\_setup\_cr, 385
- PreAMGSetupRS.c, 385
  - fasp\_amg\_setup\_rs, 386
- PreAMGSetupSA.c, 386
  - fasp\_amg\_setup\_sa, 387
- PreAMGSetupSABSR.c, 387
  - fasp\_amg\_setup\_sa\_bsr, 388
- PreAMGSetupUA.c, 389
  - fasp\_amg\_setup\_ua, 389
- PreAMGSetupUABSR.c, 390
  - fasp\_amg\_setup\_ua\_bsr, 390
- PreBLC.c, 391
  - fasp\_precond\_dblc\_diag\_3, 392
  - fasp\_precond\_dblc\_diag\_3\_amg, 392
  - fasp\_precond\_dblc\_diag\_4, 393
  - fasp\_precond\_dblc\_lower\_3, 393
  - fasp\_precond\_dblc\_lower\_3\_amg, 395
  - fasp\_precond\_dblc\_lower\_4, 395
  - fasp\_precond\_dblc\_SGS\_3, 397
  - fasp\_precond\_dblc\_SGS\_3\_amg, 397
  - fasp\_precond\_dblc\_sweeping, 399
  - fasp\_precond\_dblc\_upper\_3, 399
  - fasp\_precond\_dblc\_upper\_3\_amg, 401
- PreBSR.c, 401
  - fasp\_precond\_dbsr\_amg, 402
  - fasp\_precond\_dbsr\_amg\_nk, 403
  - fasp\_precond\_dbsr\_diag, 403
  - fasp\_precond\_dbsr\_diag\_nc2, 404
  - fasp\_precond\_dbsr\_diag\_nc3, 404
  - fasp\_precond\_dbsr\_diag\_nc5, 405
  - fasp\_precond\_dbsr\_diag\_nc7, 405
  - fasp\_precond\_dbsr\_ilu, 406
  - fasp\_precond\_dbsr\_ilu\_ls\_omp, 406
  - fasp\_precond\_dbsr\_ilu\_mc\_omp, 407
  - fasp\_precond\_dbsr\_namli, 407
- PREC\_AMG
  - fasp\_const.h, 288
- PREC\_DIAG
  - fasp\_const.h, 288
- PREC\_FMG
  - fasp\_const.h, 289
- PREC\_ILU
  - fasp\_const.h, 289
- PREC\_NULL
  - fasp\_const.h, 289
- PREC\_SCHWARZ
  - fasp\_const.h, 289
- precond, 56
- precond\_data, 56
- precond\_data\_blc, 58
  - A\_diag, 59
  - Ablc, 59
  - amgparam, 59
  - LU\_diag, 59
  - mgl, 59
  - r, 60
- precond\_data\_bsr, 60
- precond\_data\_str, 62
- precond\_data\_sweeping, 63
  - A, 64
  - Ai, 64
  - local\_A, 65
  - local\_index, 65
  - local\_LU, 65
  - NumLayers, 65
  - r, 65
  - w, 66
- precond\_diag\_bsr, 66
- precond\_diag\_str, 67
- precond\_type
  - input\_param, 49
  - ITS\_param, 52
- PreCSR.c, 408
  - fasp\_precond\_amg, 409
  - fasp\_precond\_amg\_nk, 409
  - fasp\_precond\_amli, 410
  - fasp\_precond\_diag, 410
  - fasp\_precond\_famg, 411
  - fasp\_precond\_ilu, 411
  - fasp\_precond\_ilu\_backward, 412
  - fasp\_precond\_ilu\_forward, 412
  - fasp\_precond\_namli, 413
  - fasp\_precond\_setup, 413
  - fasp\_precond\_swz, 414
- PreDataInit.c, 414
  - fasp\_amg\_data\_bsr\_create, 415
  - fasp\_amg\_data\_bsr\_free, 415
  - fasp\_amg\_data\_create, 416
  - fasp\_amg\_data\_free, 416
  - fasp\_ilu\_data\_create, 417
  - fasp\_ilu\_data\_free, 417



- fasp\_solver\_dblc\_krylov\_sweeping, [437](#)
- SolBSR.c, [438](#)
  - fasp\_solver\_dbsr\_itsolver, [439](#)
  - fasp\_solver\_dbsr\_krylov, [439](#)
  - fasp\_solver\_dbsr\_krylov\_amg, [440](#)
  - fasp\_solver\_dbsr\_krylov\_amg\_nk, [441](#)
  - fasp\_solver\_dbsr\_krylov\_diag, [441](#)
  - fasp\_solver\_dbsr\_krylov\_ilu, [442](#)
  - fasp\_solver\_dbsr\_krylov\_nk\_amg, [442](#)
- SolCSR.c, [443](#)
  - fasp\_solver\_dcsr\_itsolver, [444](#)
  - fasp\_solver\_dcsr\_itsolver\_s, [445](#)
  - fasp\_solver\_dcsr\_krylov, [446](#)
  - fasp\_solver\_dcsr\_krylov\_amg, [446](#)
  - fasp\_solver\_dcsr\_krylov\_amg\_nk, [447](#)
  - fasp\_solver\_dcsr\_krylov\_diag, [447](#)
  - fasp\_solver\_dcsr\_krylov\_ilu, [448](#)
  - fasp\_solver\_dcsr\_krylov\_ilu\_M, [449](#)
  - fasp\_solver\_dcsr\_krylov\_s, [449](#)
  - fasp\_solver\_dcsr\_krylov\_swz, [450](#)
- SolFAMG.c, [451](#)
  - fasp\_solver\_famg, [451](#)
- SolGMGPoisson.c, [452](#)
  - fasp\_poisson\_fgm1d, [452](#)
  - fasp\_poisson\_fgm2d, [453](#)
  - fasp\_poisson\_fgm3d, [454](#)
  - fasp\_poisson\_gmg1d, [454](#)
  - fasp\_poisson\_gmg2d, [456](#)
  - fasp\_poisson\_gmg3d, [457](#)
  - fasp\_poisson\_gmgcg1d, [457](#)
  - fasp\_poisson\_gmgcg2d, [458](#)
  - fasp\_poisson\_gmgcg3d, [459](#)
- SolMatFree.c, [460](#)
  - fasp\_solver\_itsolver, [460](#)
  - fasp\_solver\_krylov, [461](#)
  - fasp\_solver\_matfree\_init, [461](#)
- SolSTR.c, [462](#)
  - fasp\_solver\_dstr\_itsolver, [463](#)
  - fasp\_solver\_dstr\_krylov, [463](#)
  - fasp\_solver\_dstr\_krylov\_blockgs, [464](#)
  - fasp\_solver\_dstr\_krylov\_diag, [465](#)
  - fasp\_solver\_dstr\_krylov\_ilu, [465](#)
- SOLVER\_AMG
  - fasp\_const.h, [292](#)
- SOLVER\_BiCGstab
  - fasp\_const.h, [292](#)
- SOLVER\_CG
  - fasp\_const.h, [292](#)
- SOLVER\_DEFAULT
  - fasp\_const.h, [293](#)
- SOLVER\_FMG
  - fasp\_const.h, [293](#)
- SOLVER\_GCG
  - fasp\_const.h, [293](#)
- SOLVER\_GCR
  - fasp\_const.h, [293](#)
- SOLVER\_GMRES
  - fasp\_const.h, [293](#)
- SOLVER\_MinRes
  - fasp\_const.h, [293](#)
- SOLVER\_MUMPS
  - fasp\_const.h, [293](#)
- SOLVER\_PARDISO
  - fasp\_const.h, [293](#)
- SOLVER\_SBiCGstab
  - fasp\_const.h, [294](#)
- SOLVER\_SCG
  - fasp\_const.h, [294](#)
- SOLVER\_SGCG
  - fasp\_const.h, [294](#)
- SOLVER\_SGMRES
  - fasp\_const.h, [294](#)
- SOLVER\_SMinRes
  - fasp\_const.h, [294](#)
- SOLVER\_SUPERLU
  - fasp\_const.h, [294](#)
- SOLVER\_SVFGMRES
  - fasp\_const.h, [294](#)
- SOLVER\_SVGMRES
  - fasp\_const.h, [294](#)
- solver\_type
  - input\_param, [49](#)
- SOLVER\_UMFPACK
  - fasp\_const.h, [295](#)
- SOLVER\_VFGMRES
  - fasp\_const.h, [295](#)
- SOLVER\_VGMRES
  - fasp\_const.h, [295](#)
- SolWrapper.c, [466](#)
  - fasp\_fwrapper\_dcsr\_amg\_, [467](#)
  - fasp\_fwrapper\_dcsr\_krylov\_amg\_, [467](#)
  - fasp\_fwrapper\_dcsr\_krylov\_ilu\_, [468](#)
  - fasp\_fwrapper\_dcsr\_pardiso\_, [469](#)
- SPAIR
  - fasp\_const.h, [295](#)
- STAG\_RATIO
  - fasp\_const.h, [295](#)
- STOP\_MOD\_REL\_RES
  - fasp\_const.h, [295](#)
- STOP\_REL\_PRECRES
  - fasp\_const.h, [295](#)
- STOP\_REL\_RES
  - fasp\_const.h, [295](#)
- stop\_type
  - input\_param, [50](#)
  - ITS\_param, [53](#)
- STRLEN
  - fasp.h, [271](#)

SWAP  
     BlaSmallMatInv.c, [183](#)  
 SWZ\_bksolver  
     input\_param, [50](#)  
 SWZ\_data, [67](#)  
 SWZ\_maxlvl  
     input\_param, [50](#)  
 SWZ\_mmsize  
     input\_param, [50](#)  
 SWZ\_param, [69](#)  
 SWZ\_type  
     input\_param, [50](#)  
  
 t  
     grid2d, [32](#)  
 tfather  
     grid2d, [33](#)  
 THDs\_AMG\_GS  
     AuxThreads.c, [105](#)  
 THDs\_CPR\_gGS  
     AuxThreads.c, [105](#)  
 THDs\_CPR\_IGS  
     AuxThreads.c, [105](#)  
 tol  
     ITS\_param, [53](#)  
 triangles  
     grid2d, [33](#)  
 TRUE  
     fasp\_const.h, [296](#)  
  
 UA\_AMG  
     fasp\_const.h, [296](#)  
 UNPT  
     fasp\_const.h, [296](#)  
 USERDEFINED  
     fasp\_const.h, [296](#)  
  
 V\_CYCLE  
     fasp\_const.h, [296](#)  
 val  
     dBSRmat, [25](#)  
 vertices  
     grid2d, [33](#)  
 VMB  
     fasp\_const.h, [296](#)  
 VW\_CYCLE  
     fasp\_const.h, [296](#)  
  
 w  
     precond\_data\_sweeping, [66](#)  
 W\_CYCLE  
     fasp\_const.h, [297](#)  
 workdir  
     input\_param, [51](#)  
 WV\_CYCLE  
     fasp\_const.h, [297](#)  
  
 XtrMumps.c, [470](#)  
     fasp\_solver\_mumps, [470](#)  
     fasp\_solver\_mumps\_steps, [472](#)  
     ICNTL, [470](#)  
 XtrPardiso.c, [472](#)  
     fasp\_solver\_pardiso, [473](#)  
 XtrSamg.c, [473](#)  
     dCSRmat2SAMGInput, [474](#)  
     dvector2SAMGInput, [474](#)  
 XtrSuperlu.c, [475](#)  
     fasp\_solver\_superlu, [475](#)  
 XtrUmfpack.c, [476](#)  
     fasp\_solver\_umfpack, [476](#)