

## Fast Auxiliary Space Preconditioning

1.9.5 Jan/28/2017

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>How to obtain FASP</b>	<b>3</b>
<b>3</b>	<b>Building and Installation</b>	<b>5</b>
<b>4</b>	<b>Developers</b>	<b>7</b>
<b>5</b>	<b>Doxygen</b>	<b>9</b>
<b>6</b>	<b>Data Structure Index</b>	<b>11</b>
6.1	Data Structures . . . . .	11
<b>7</b>	<b>File Index</b>	<b>13</b>
7.1	File List . . . . .	13
<b>8</b>	<b>Data Structure Documentation</b>	<b>19</b>
8.1	AMG_data Struct Reference . . . . .	19
8.1.1	Detailed Description . . . . .	20
8.2	AMG_data_bsr Struct Reference . . . . .	20
8.2.1	Detailed Description . . . . .	22
8.3	AMG_param Struct Reference . . . . .	22
8.3.1	Detailed Description . . . . .	24
8.4	block_dvector Struct Reference . . . . .	24
8.4.1	Detailed Description . . . . .	25

8.5	block_ivector Struct Reference	25
8.5.1	Detailed Description	25
8.6	dBLCmat Struct Reference	26
8.6.1	Detailed Description	26
8.7	dBSRmat Struct Reference	26
8.7.1	Detailed Description	27
8.7.2	Field Documentation	27
8.7.2.1	JA	27
8.7.2.2	val	27
8.8	dCOOmat Struct Reference	27
8.8.1	Detailed Description	28
8.9	dCSRLmat Struct Reference	28
8.9.1	Detailed Description	29
8.10	dCSRmat Struct Reference	29
8.10.1	Detailed Description	30
8.11	ddenmat Struct Reference	30
8.11.1	Detailed Description	30
8.12	dSTRmat Struct Reference	31
8.12.1	Detailed Description	31
8.13	dvector Struct Reference	32
8.13.1	Detailed Description	32
8.14	grid2d Struct Reference	32
8.14.1	Detailed Description	33
8.14.2	Field Documentation	33
8.14.2.1	e	33
8.14.2.2	edges	33
8.14.2.3	ediri	33
8.14.2.4	efather	34

8.14.2.5	p	34
8.14.2.6	pdir	34
8.14.2.7	pfather	34
8.14.2.8	s	34
8.14.2.9	t	35
8.14.2.10	tfather	35
8.14.2.11	triangles	35
8.14.2.12	vertices	35
8.15	iBLCmat Struct Reference	35
8.15.1	Detailed Description	36
8.16	iCOOmat Struct Reference	36
8.16.1	Detailed Description	37
8.17	iCSRmat Struct Reference	37
8.17.1	Detailed Description	37
8.18	idenmat Struct Reference	38
8.18.1	Detailed Description	38
8.19	ILU_data Struct Reference	38
8.19.1	Detailed Description	39
8.20	ILU_param Struct Reference	40
8.20.1	Detailed Description	40
8.21	input_param Struct Reference	40
8.21.1	Detailed Description	42
8.21.2	Field Documentation	42
8.21.2.1	AMG_aggregation_type	42
8.21.2.2	AMG_aggressive_level	42
8.21.2.3	AMG_aggressive_path	42
8.21.2.4	AMG_amli_degree	43
8.21.2.5	AMG_coarse_dof	43

8.21.2.6	AMG_coarse_scaling	43
8.21.2.7	AMG_coarse_solver	43
8.21.2.8	AMG_coarsening_type	43
8.21.2.9	AMG_cycle_type	44
8.21.2.10	AMG_ILU_levels	44
8.21.2.11	AMG_interpolation_type	44
8.21.2.12	AMG_levels	44
8.21.2.13	AMG_max_aggregation	44
8.21.2.14	AMG_max_row_sum	45
8.21.2.15	AMG_maxit	45
8.21.2.16	AMG_nl_amli_krylov_type	45
8.21.2.17	AMG_pair_number	45
8.21.2.18	AMG_polynomial_degree	45
8.21.2.19	AMG_postsmooth_iter	46
8.21.2.20	AMG_presmooth_iter	46
8.21.2.21	AMG_quality_bound	46
8.21.2.22	AMG_relaxation	46
8.21.2.23	AMG_Schwarz_levels	46
8.21.2.24	AMG_smooth_filter	47
8.21.2.25	AMG_smooth_order	47
8.21.2.26	AMG_smoother	47
8.21.2.27	AMG_strong_coupled	47
8.21.2.28	AMG_strong_threshold	47
8.21.2.29	AMG_tentative_smooth	48
8.21.2.30	AMG_tol	48
8.21.2.31	AMG_truncation_threshold	48
8.21.2.32	AMG_type	48
8.21.2.33	ILU_droptol	48

8.21.2.34 ILU_lfil . . . . .	49
8.21.2.35 ILU_permtol . . . . .	49
8.21.2.36 ILU_relax . . . . .	49
8.21.2.37 ILU_type . . . . .	49
8.21.2.38 inifile . . . . .	49
8.21.2.39 itsolver_maxit . . . . .	50
8.21.2.40 itsolver_tol . . . . .	50
8.21.2.41 output_type . . . . .	50
8.21.2.42 precondition_type . . . . .	50
8.21.2.43 print_level . . . . .	50
8.21.2.44 problem_num . . . . .	51
8.21.2.45 restart . . . . .	51
8.21.2.46 Schwarz_blksolver . . . . .	51
8.21.2.47 Schwarz_maxlvl . . . . .	51
8.21.2.48 Schwarz_mmsize . . . . .	51
8.21.2.49 Schwarz_type . . . . .	52
8.21.2.50 solver_type . . . . .	52
8.21.2.51 stop_type . . . . .	52
8.21.2.52 workdir . . . . .	52
8.22 itsolver_param Struct Reference . . . . .	52
8.22.1 Detailed Description . . . . .	53
8.22.2 Field Documentation . . . . .	53
8.22.2.1 itsolver_type . . . . .	53
8.22.2.2 maxit . . . . .	53
8.22.2.3 precondition_type . . . . .	53
8.22.2.4 print_level . . . . .	54
8.22.2.5 restart . . . . .	54
8.22.2.6 stop_type . . . . .	54

8.22.2.7	tol	54
8.23	ivector Struct Reference	54
8.23.1	Detailed Description	55
8.24	mallinfo Struct Reference	55
8.24.1	Detailed Description	55
8.25	malloc_chunk Struct Reference	56
8.25.1	Detailed Description	56
8.26	malloc_params Struct Reference	56
8.26.1	Detailed Description	56
8.27	malloc_segment Struct Reference	56
8.27.1	Detailed Description	57
8.28	malloc_state Struct Reference	57
8.28.1	Detailed Description	57
8.29	malloc_tree_chunk Struct Reference	58
8.29.1	Detailed Description	58
8.30	Mumps_data Struct Reference	58
8.30.1	Detailed Description	58
8.31	mxv_matfree Struct Reference	59
8.31.1	Detailed Description	59
8.32	nedmallinfo Struct Reference	59
8.32.1	Detailed Description	59
8.33	Pardiso_data Struct Reference	60
8.33.1	Detailed Description	60
8.34	precond Struct Reference	60
8.34.1	Detailed Description	60
8.35	precond_block_data Struct Reference	61
8.35.1	Detailed Description	61
8.35.2	Field Documentation	61



8.35.2.1	A_diag	61
8.35.2.2	Ablc	61
8.35.2.3	amgparam	62
8.35.2.4	LU_diag	62
8.35.2.5	mgl	62
8.35.2.6	r	62
8.36	precond_data Struct Reference	62
8.36.1	Detailed Description	64
8.37	precond_data_bsr Struct Reference	64
8.37.1	Detailed Description	65
8.38	precond_data_str Struct Reference	66
8.38.1	Detailed Description	67
8.39	precond_diagbsr Struct Reference	67
8.39.1	Detailed Description	68
8.40	precond_diagstr Struct Reference	68
8.40.1	Detailed Description	68
8.41	precond_sweeping_data Struct Reference	69
8.41.1	Detailed Description	69
8.41.2	Field Documentation	69
8.41.2.1	A	69
8.41.2.2	Ai	70
8.41.2.3	local_A	70
8.41.2.4	local_index	70
8.41.2.5	local_LU	70
8.41.2.6	NumLayers	70
8.41.2.7	r	71
8.41.2.8	w	71
8.42	Schwarz_data Struct Reference	71
8.42.1	Detailed Description	72
8.43	Schwarz_param Struct Reference	72
8.43.1	Detailed Description	73

<b>9 File Documentation</b>	<b>75</b>
9.1 AuxArray.c File Reference	75
9.1.1 Detailed Description	76
9.1.2 Function Documentation	76
9.1.2.1 fasp_array_cp()	76
9.1.2.2 fasp_array_cp_nc3()	76
9.1.2.3 fasp_array_cp_nc5()	77
9.1.2.4 fasp_array_cp_nc7()	78
9.1.2.5 fasp_array_null()	78
9.1.2.6 fasp_array_set()	79
9.1.2.7 fasp_iarray_cp()	79
9.1.2.8 fasp_iarray_set()	80
9.2 AuxConvert.c File Reference	81
9.2.1 Detailed Description	81
9.2.2 Function Documentation	81
9.2.2.1 fasp_aux_bbyteToldouble()	81
9.2.2.2 fasp_aux_change_endian4()	82
9.2.2.3 fasp_aux_change_endian8()	82
9.3 AuxGivens.c File Reference	83
9.3.1 Detailed Description	83
9.3.2 Function Documentation	83
9.3.2.1 fasp_aux_givens()	83
9.4 AuxGraphics.c File Reference	84
9.4.1 Detailed Description	84
9.4.2 Function Documentation	85
9.4.2.1 fasp_dbsr_plot()	85
9.4.2.2 fasp_dbsr_subplot()	85
9.4.2.3 fasp_dcsr_plot()	86

9.4.2.4	<a href="#">fasp_dcsr_subplot()</a>	87
9.4.2.5	<a href="#">fasp_grid2d_plot()</a>	88
9.5	<a href="#">AuxInput.c File Reference</a>	88
9.5.1	<a href="#">Detailed Description</a>	88
9.5.2	<a href="#">Function Documentation</a>	89
9.5.2.1	<a href="#">fasp_param_check()</a>	89
9.5.2.2	<a href="#">fasp_param_input()</a>	89
9.6	<a href="#">AuxMemory.c File Reference</a>	90
9.6.1	<a href="#">Detailed Description</a>	91
9.6.2	<a href="#">Function Documentation</a>	91
9.6.2.1	<a href="#">fasp_mem_calloc()</a>	91
9.6.2.2	<a href="#">fasp_mem_check()</a>	92
9.6.2.3	<a href="#">fasp_mem_free()</a>	92
9.6.2.4	<a href="#">fasp_mem_iludata_check()</a>	93
9.6.2.5	<a href="#">fasp_mem_realloc()</a>	93
9.6.2.6	<a href="#">fasp_mem_usage()</a>	94
9.6.3	<a href="#">Variable Documentation</a>	94
9.6.3.1	<a href="#">total_alloc_count</a>	95
9.6.3.2	<a href="#">total_alloc_mem</a>	95
9.7	<a href="#">AuxMessage.c File Reference</a>	95
9.7.1	<a href="#">Detailed Description</a>	96
9.7.2	<a href="#">Function Documentation</a>	96
9.7.2.1	<a href="#">fasp_chkerr()</a>	96
9.7.2.2	<a href="#">print_amgcomplexity()</a>	96
9.7.2.3	<a href="#">print_amgcomplexity_bsr()</a>	97
9.7.2.4	<a href="#">print_cputime()</a>	97
9.7.2.5	<a href="#">print_itinfo()</a>	98
9.7.2.6	<a href="#">print_message()</a>	99

9.8	AuxParam.c File Reference	99
9.8.1	Detailed Description	100
9.8.2	Function Documentation	101
9.8.2.1	fasp_param_amg_init()	101
9.8.2.2	fasp_param_amg_print()	101
9.8.2.3	fasp_param_amg_set()	102
9.8.2.4	fasp_param_amg_to_prec()	102
9.8.2.5	fasp_param_amg_to_prec_bsr()	103
9.8.2.6	fasp_param_ilu_init()	103
9.8.2.7	fasp_param_ilu_print()	104
9.8.2.8	fasp_param_ilu_set()	104
9.8.2.9	fasp_param_init()	105
9.8.2.10	fasp_param_input_init()	106
9.8.2.11	fasp_param_prec_to_amg()	106
9.8.2.12	fasp_param_prec_to_amg_bsr()	107
9.8.2.13	fasp_param_schwarz_init()	107
9.8.2.14	fasp_param_schwarz_print()	108
9.8.2.15	fasp_param_schwarz_set()	108
9.8.2.16	fasp_param_set()	109
9.8.2.17	fasp_param_solver_init()	109
9.8.2.18	fasp_param_solver_print()	110
9.8.2.19	fasp_param_solver_set()	110
9.9	AuxSort.c File Reference	111
9.9.1	Detailed Description	111
9.9.2	Function Documentation	111
9.9.2.1	fasp_aux_dQuickSort()	111
9.9.2.2	fasp_aux_dQuickSortIndex()	112
9.9.2.3	fasp_aux_iQuickSort()	113

9.9.2.4	<a href="#">fasp_aux_iQuickSortIndex()</a>	114
9.9.2.5	<a href="#">fasp_aux_merge()</a>	115
9.9.2.6	<a href="#">fasp_aux_msort()</a>	116
9.9.2.7	<a href="#">fasp_aux_unique()</a>	117
9.9.2.8	<a href="#">fasp_BinarySearch()</a>	117
9.9.2.9	<a href="#">fasp_multicolors_independent_set()</a>	118
9.9.2.10	<a href="#">fasp_topological_sorting_ilu()</a>	118
9.10	<a href="#">AuxThreads.c File Reference</a>	119
9.10.1	<a href="#">Detailed Description</a>	119
9.10.2	<a href="#">Function Documentation</a>	120
9.10.2.1	<a href="#">fasp_get_start_end()</a>	120
9.10.2.2	<a href="#">fasp_set_GS_threads()</a>	120
9.10.3	<a href="#">Variable Documentation</a>	121
9.10.3.1	<a href="#">THDs_AMG_GS</a>	121
9.10.3.2	<a href="#">THDs_CPR_gGS</a>	121
9.10.3.3	<a href="#">THDs_CPR_IGS</a>	121
9.11	<a href="#">AuxTiming.c File Reference</a>	122
9.11.1	<a href="#">Detailed Description</a>	122
9.11.2	<a href="#">Function Documentation</a>	122
9.11.2.1	<a href="#">fasp_gettime()</a>	122
9.12	<a href="#">AuxVector.c File Reference</a>	123
9.12.1	<a href="#">Detailed Description</a>	123
9.12.2	<a href="#">Function Documentation</a>	124
9.12.2.1	<a href="#">fasp_dvec_alloc()</a>	124
9.12.2.2	<a href="#">fasp_dvec_cp()</a>	124
9.12.2.3	<a href="#">fasp_dvec_create()</a>	125
9.12.2.4	<a href="#">fasp_dvec_free()</a>	125
9.12.2.5	<a href="#">fasp_dvec_isnan()</a>	126

9.12.2.6	<a href="#">fasp_dvec_maxdiff()</a>	126
9.12.2.7	<a href="#">fasp_dvec_null()</a>	127
9.12.2.8	<a href="#">fasp_dvec_rand()</a>	128
9.12.2.9	<a href="#">fasp_dvec_set()</a>	129
9.12.2.10	<a href="#">fasp_dvec_symdiagscale()</a>	129
9.12.2.11	<a href="#">fasp_ivec_alloc()</a>	130
9.12.2.12	<a href="#">fasp_ivec_create()</a>	130
9.12.2.13	<a href="#">fasp_ivec_free()</a>	131
9.12.2.14	<a href="#">fasp_ivec_set()</a>	131
9.13	<a href="#">BlaArray.c File Reference</a>	132
9.13.1	<a href="#">Detailed Description</a>	133
9.13.2	<a href="#">Function Documentation</a>	133
9.13.2.1	<a href="#">fasp_blas_array_ax()</a>	133
9.13.2.2	<a href="#">fasp_blas_array_axpby()</a>	134
9.13.2.3	<a href="#">fasp_blas_array_axpy()</a>	134
9.13.2.4	<a href="#">fasp_blas_array_axpyz()</a>	135
9.13.2.5	<a href="#">fasp_blas_array_dotprod()</a>	136
9.13.2.6	<a href="#">fasp_blas_array_norm1()</a>	137
9.13.2.7	<a href="#">fasp_blas_array_norm2()</a>	137
9.13.2.8	<a href="#">fasp_blas_array_norminf()</a>	138
9.14	<a href="#">BlaEigen.c File Reference</a>	139
9.14.1	<a href="#">Detailed Description</a>	139
9.14.2	<a href="#">Function Documentation</a>	139
9.14.2.1	<a href="#">fasp_dcsr_eig()</a>	139
9.15	<a href="#">BlaFormat.c File Reference</a>	140
9.15.1	<a href="#">Detailed Description</a>	141
9.15.2	<a href="#">Function Documentation</a>	141
9.15.2.1	<a href="#">fasp_format_dblc_dcsr()</a>	141

9.15.2.2	<a href="#">fasp_format_dbsr_dcoo()</a>	141
9.15.2.3	<a href="#">fasp_format_dbsr_dcsr()</a>	142
9.15.2.4	<a href="#">fasp_format_dcoo_dcsr()</a>	143
9.15.2.5	<a href="#">fasp_format_dcsr_dbsr()</a>	143
9.15.2.6	<a href="#">fasp_format_dcsr_dcoo()</a>	144
9.15.2.7	<a href="#">fasp_format_dcsr_dcsr()</a>	145
9.15.2.8	<a href="#">fasp_format_dstr_dbsr()</a>	145
9.15.2.9	<a href="#">fasp_format_dstr_dcsr()</a>	146
9.16	<a href="#">BlalLU.c File Reference</a>	147
9.16.1	<a href="#">Detailed Description</a>	147
9.16.2	<a href="#">Function Documentation</a>	147
9.16.2.1	<a href="#">fasp_iluk()</a>	148
9.16.2.2	<a href="#">fasp_ilut()</a>	149
9.16.2.3	<a href="#">fasp_ilutp()</a>	150
9.16.2.4	<a href="#">fasp_symbfactor()</a>	152
9.17	<a href="#">BlalLUSetupBSR.c File Reference</a>	154
9.17.1	<a href="#">Detailed Description</a>	155
9.17.2	<a href="#">Function Documentation</a>	155
9.17.2.1	<a href="#">fasp_ilu_dbsr_setup()</a>	155
9.17.2.2	<a href="#">fasp_ilu_dbsr_setup_levsch_omp()</a>	156
9.17.2.3	<a href="#">fasp_ilu_dbsr_setup_mc_omp()</a>	157
9.17.2.4	<a href="#">fasp_ilu_dbsr_setup_omp()</a>	157
9.18	<a href="#">BlalLUSetupCSR.c File Reference</a>	158
9.18.1	<a href="#">Detailed Description</a>	158
9.18.2	<a href="#">Function Documentation</a>	159
9.18.2.1	<a href="#">fasp_ilu_dcsr_setup()</a>	159
9.19	<a href="#">BlalLUSetupSTR.c File Reference</a>	159
9.19.1	<a href="#">Detailed Description</a>	160

9.19.2	Function Documentation	160
9.19.2.1	fasp_ilu_dstr_setup0()	160
9.19.2.2	fasp_ilu_dstr_setup1()	161
9.20	BlaIO.c File Reference	161
9.20.1	Detailed Description	163
9.20.2	Function Documentation	163
9.20.2.1	fasp_dbsr_print()	164
9.20.2.2	fasp_dbsr_read()	164
9.20.2.3	fasp_dbsr_write()	165
9.20.2.4	fasp_dbsr_write_coo()	166
9.20.2.5	fasp_dcoo1_read()	166
9.20.2.6	fasp_dcoo_print()	167
9.20.2.7	fasp_dcoo_read()	167
9.20.2.8	fasp_dcoo_shift_read()	168
9.20.2.9	fasp_dcoo_write()	169
9.20.2.10	fasp_dcsr_print()	169
9.20.2.11	fasp_dcsr_read()	170
9.20.2.12	fasp_dcsr_write_coo()	170
9.20.2.13	fasp_dcsrvec1_read()	171
9.20.2.14	fasp_dcsrvec1_write()	172
9.20.2.15	fasp_dcsrvec2_read()	173
9.20.2.16	fasp_dcsrvec2_write()	174
9.20.2.17	fasp_dmtx_read()	175
9.20.2.18	fasp_dmtxsym_read()	175
9.20.2.19	fasp_dstr_print()	176
9.20.2.20	fasp_dstr_read()	176
9.20.2.21	fasp_dstr_write()	177
9.20.2.22	fasp_dvec_print()	178



9.20.2.23 fasp_dvec_read()	178
9.20.2.24 fasp_dvec_write()	179
9.20.2.25 fasp_dvecind_read()	180
9.20.2.26 fasp_dvecind_write()	180
9.20.2.27 fasp_hb_read()	181
9.20.2.28 fasp_ivec_print()	182
9.20.2.29 fasp_ivec_read()	182
9.20.2.30 fasp_ivec_write()	183
9.20.2.31 fasp_ivecind_read()	183
9.20.2.32 fasp_matrix_read()	184
9.20.2.33 fasp_matrix_read_bin()	185
9.20.2.34 fasp_matrix_write()	186
9.20.2.35 fasp_vector_read()	186
9.20.2.36 fasp_vector_write()	187
9.20.3 Variable Documentation	188
9.20.3.1 dlength	188
9.20.3.2 ilength	188
9.21 BlaOrderingCSR.c File Reference	189
9.21.1 Detailed Description	189
9.21.2 Function Documentation	189
9.21.2.1 fasp_dcsr_CMK_order()	189
9.21.2.2 fasp_dcsr_RCMK_order()	190
9.22 BlaSchwarzSetup.c File Reference	190
9.22.1 Detailed Description	191
9.22.2 Function Documentation	191
9.22.2.1 fasp_dcsr_schwarz_backward_smoother()	191
9.22.2.2 fasp_dcsr_schwarz_forward_smoother()	192
9.22.2.3 fasp_schwarz_setup()	192

9.23 BlaSmallMat.c File Reference . . . . .	193
9.23.1 Detailed Description . . . . .	195
9.23.2 Function Documentation . . . . .	195
9.23.2.1 fasp_blas_array_axpy_nc2() . . . . .	195
9.23.2.2 fasp_blas_array_axpy_nc3() . . . . .	196
9.23.2.3 fasp_blas_array_axpy_nc5() . . . . .	196
9.23.2.4 fasp_blas_array_axpy_nc7() . . . . .	197
9.23.2.5 fasp_blas_array_axpyz_nc2() . . . . .	197
9.23.2.6 fasp_blas_array_axpyz_nc3() . . . . .	198
9.23.2.7 fasp_blas_array_axpyz_nc5() . . . . .	199
9.23.2.8 fasp_blas_array_axpyz_nc7() . . . . .	200
9.23.2.9 fasp_blas_smat_aAxpby() . . . . .	200
9.23.2.10 fasp_blas_smat_add() . . . . .	201
9.23.2.11 fasp_blas_smat_axm() . . . . .	202
9.23.2.12 fasp_blas_smat_mul() . . . . .	202
9.23.2.13 fasp_blas_smat_mul_nc2() . . . . .	203
9.23.2.14 fasp_blas_smat_mul_nc3() . . . . .	204
9.23.2.15 fasp_blas_smat_mul_nc5() . . . . .	204
9.23.2.16 fasp_blas_smat_mul_nc7() . . . . .	205
9.23.2.17 fasp_blas_smat_m xv() . . . . .	205
9.23.2.18 fasp_blas_smat_m xv_nc2() . . . . .	206
9.23.2.19 fasp_blas_smat_m xv_nc3() . . . . .	207
9.23.2.20 fasp_blas_smat_m xv_nc5() . . . . .	207
9.23.2.21 fasp_blas_smat_m xv_nc7() . . . . .	208
9.23.2.22 fasp_blas_smat_ymAx() . . . . .	208
9.23.2.23 fasp_blas_smat_ymAx_nc2() . . . . .	209
9.23.2.24 fasp_blas_smat_ymAx_nc3() . . . . .	210
9.23.2.25 fasp_blas_smat_ymAx_nc5() . . . . .	210

9.23.2.26 fasp_blas_smat_ymAx_nc7()	211
9.23.2.27 fasp_blas_smat_ypAx()	212
9.23.2.28 fasp_blas_smat_ypAx_nc2()	212
9.23.2.29 fasp_blas_smat_ypAx_nc3()	213
9.23.2.30 fasp_blas_smat_ypAx_nc5()	214
9.23.2.31 fasp_blas_smat_ypAx_nc7()	214
9.24 BlaSmallMatInv.c File Reference	215
9.24.1 Detailed Description	216
9.24.2 Macro Definition Documentation	216
9.24.2.1 SWAP	216
9.24.3 Function Documentation	216
9.24.3.1 fasp_smat_identity()	216
9.24.3.2 fasp_smat_identity_nc2()	217
9.24.3.3 fasp_smat_identity_nc3()	217
9.24.3.4 fasp_smat_identity_nc5()	218
9.24.3.5 fasp_smat_identity_nc7()	218
9.24.3.6 fasp_smat_inv()	219
9.24.3.7 fasp_smat_inv_nc()	219
9.24.3.8 fasp_smat_inv_nc2()	220
9.24.3.9 fasp_smat_inv_nc3()	220
9.24.3.10 fasp_smat_inv_nc4()	221
9.24.3.11 fasp_smat_inv_nc5()	221
9.24.3.12 fasp_smat_inv_nc7()	222
9.24.3.13 fasp_smat_invp_nc()	222
9.24.3.14 fasp_smat_Linfinity()	223
9.25 BlaSmallMatLU.c File Reference	224
9.25.1 Detailed Description	224
9.25.2 Function Documentation	224

9.25.2.1	<code>fasp_smat_lu_decomp()</code>	224
9.25.2.2	<code>fasp_smat_lu_solve()</code>	225
9.26	BlaSparseBLC.c File Reference	226
9.26.1	Detailed Description	226
9.26.2	Function Documentation	227
9.26.2.1	<code>fasp_dbic_free()</code>	227
9.26.2.2	<code>fasp_dbsr_getblk()</code>	227
9.27	BlaSparseBSR.c File Reference	228
9.27.1	Detailed Description	229
9.27.2	Function Documentation	229
9.27.2.1	<code>fasp_dbsr_alloc()</code>	229
9.27.2.2	<code>fasp_dbsr_cp()</code>	230
9.27.2.3	<code>fasp_dbsr_create()</code>	230
9.27.2.4	<code>fasp_dbsr_diaginv()</code>	231
9.27.2.5	<code>fasp_dbsr_diaginv2()</code>	232
9.27.2.6	<code>fasp_dbsr_diaginv3()</code>	232
9.27.2.7	<code>fasp_dbsr_diaginv4()</code>	233
9.27.2.8	<code>fasp_dbsr_diagLU()</code>	234
9.27.2.9	<code>fasp_dbsr_diagLU2()</code>	235
9.27.2.10	<code>fasp_dbsr_diagpref()</code>	235
9.27.2.11	<code>fasp_dbsr_free()</code>	236
9.27.2.12	<code>fasp_dbsr_getdiag()</code>	237
9.27.2.13	<code>fasp_dbsr_getdiaginv()</code>	237
9.27.2.14	<code>fasp_dbsr_null()</code>	238
9.27.2.15	<code>fasp_dbsr_perm()</code>	238
9.27.2.16	<code>fasp_dbsr_trans()</code>	239
9.28	BlaSparseCheck.c File Reference	240
9.28.1	Detailed Description	240

9.28.2	Function Documentation	240
9.28.2.1	<code>fasp_check_dCSRmat()</code>	240
9.28.2.2	<code>fasp_check_diagdom()</code>	241
9.28.2.3	<code>fasp_check_diagpos()</code>	242
9.28.2.4	<code>fasp_check_diagzero()</code>	242
9.28.2.5	<code>fasp_check_iCSRmat()</code>	243
9.28.2.6	<code>fasp_check_symm()</code>	243
9.29	BlaSparseCOO.c File Reference	244
9.29.1	Detailed Description	244
9.29.2	Function Documentation	245
9.29.2.1	<code>fasp_dcoo_alloc()</code>	245
9.29.2.2	<code>fasp_dcoo_create()</code>	245
9.29.2.3	<code>fasp_dcoo_free()</code>	246
9.29.2.4	<code>fasp_dcoo_shift()</code>	246
9.30	BlaSparseCSR.c File Reference	247
9.30.1	Detailed Description	249
9.30.2	Function Documentation	249
9.30.2.1	<code>fasp_dcsr_alloc()</code>	249
9.30.2.2	<code>fasp_dcsr_bandwidth()</code>	249
9.30.2.3	<code>fasp_dcsr_compress()</code>	250
9.30.2.4	<code>fasp_dcsr_compress_inplace()</code>	251
9.30.2.5	<code>fasp_dcsr_cp()</code>	251
9.30.2.6	<code>fasp_dcsr_create()</code>	252
9.30.2.7	<code>fasp_dcsr_diagpref()</code>	252
9.30.2.8	<code>fasp_dcsr_free()</code>	253
9.30.2.9	<code>fasp_dcsr_getblk()</code>	254
9.30.2.10	<code>fasp_dcsr_getcol()</code>	255
9.30.2.11	<code>fasp_dcsr_getdiag()</code>	255

9.30.2.12 fasp_dcsr_multicoloring()	256
9.30.2.13 fasp_dcsr_null()	256
9.30.2.14 fasp_dcsr_perm()	257
9.30.2.15 fasp_dcsr_permz()	258
9.30.2.16 fasp_dcsr_regdiag()	258
9.30.2.17 fasp_dcsr_shift()	259
9.30.2.18 fasp_dcsr_sort()	259
9.30.2.19 fasp_dcsr_sortz()	260
9.30.2.20 fasp_dcsr_symdiagscale()	261
9.30.2.21 fasp_dcsr_sympart()	261
9.30.2.22 fasp_dcsr_trans()	262
9.30.2.23 fasp_dcsr_transz()	262
9.30.2.24 fasp_icsr_cp()	263
9.30.2.25 fasp_icsr_create()	264
9.30.2.26 fasp_icsr_free()	264
9.30.2.27 fasp_icsr_null()	265
9.30.2.28 fasp_icsr_trans()	265
9.31 BlaSparseCSRL.c File Reference	266
9.31.1 Detailed Description	266
9.31.2 Function Documentation	266
9.31.2.1 fasp_dcsrl_create()	266
9.31.2.2 fasp_dcsrl_free()	267
9.32 BlaSparseSTR.c File Reference	267
9.32.1 Detailed Description	268
9.32.2 Function Documentation	268
9.32.2.1 fasp_dstr_alloc()	268
9.32.2.2 fasp_dstr_cp()	269
9.32.2.3 fasp_dstr_create()	269

9.32.2.4	<code>fasp_dstr_free()</code>	270
9.32.2.5	<code>fasp_dstr_null()</code>	271
9.33	BlaSparseUtil.c File Reference	271
9.33.1	Detailed Description	272
9.33.2	Function Documentation	272
9.33.2.1	<code>fasp_sparse_aat()</code>	273
9.33.2.2	<code>fasp_sparse_abyb()</code>	273
9.33.2.3	<code>fasp_sparse_abybms()</code>	274
9.33.2.4	<code>fasp_sparse_aplbms()</code>	275
9.33.2.5	<code>fasp_sparse_aplusb()</code>	275
9.33.2.6	<code>fasp_sparse_iit()</code>	276
9.33.2.7	<code>fasp_sparse_MIS()</code>	277
9.33.2.8	<code>fasp_sparse_rapcmp()</code>	277
9.33.2.9	<code>fasp_sparse_rapms()</code>	278
9.33.2.10	<code>fasp_sparse_wta()</code>	279
9.33.2.11	<code>fasp_sparse_wtams()</code>	280
9.33.2.12	<code>fasp_sparse_ytx()</code>	281
9.33.2.13	<code>fasp_sparse_ytxbig()</code>	281
9.34	BlaSpmvBLC.c File Reference	282
9.34.1	Detailed Description	282
9.34.2	Function Documentation	282
9.34.2.1	<code>fasp_blas_dblc_aApy()</code>	282
9.34.2.2	<code>fasp_blas_dblc_mxv()</code>	283
9.35	BlaSpmvBSR.c File Reference	284
9.35.1	Detailed Description	284
9.35.2	Function Documentation	284
9.35.2.1	<code>fasp_blas_dbsr_aApyby()</code>	285
9.35.2.2	<code>fasp_blas_dbsr_aApy()</code>	285

9.35.2.3	<code>fasp_blas_dbsr_aApy_agg()</code>	286
9.35.2.4	<code>fasp_blas_dbsr_axm()</code>	287
9.35.2.5	<code>fasp_blas_dbsr_mxm()</code>	287
9.35.2.6	<code>fasp_blas_dbsr_m xv()</code>	288
9.35.2.7	<code>fasp_blas_dbsr_m xv_agg()</code>	289
9.35.2.8	<code>fasp_blas_dbsr_rap()</code>	289
9.35.2.9	<code>fasp_blas_dbsr_rap1()</code>	290
9.35.2.10	<code>fasp_blas_dbsr_rap_agg()</code>	291
9.36	BlaSpmvCSR.c File Reference	291
9.36.1	Detailed Description	292
9.36.2	Function Documentation	293
9.36.2.1	<code>fasp_blas_dcsr_aApy()</code>	293
9.36.2.2	<code>fasp_blas_dcsr_aApy_agg()</code>	293
9.36.2.3	<code>fasp_blas_dcsr_add()</code>	294
9.36.2.4	<code>fasp_blas_dcsr_axm()</code>	295
9.36.2.5	<code>fasp_blas_dcsr_mxm()</code>	295
9.36.2.6	<code>fasp_blas_dcsr_m xv()</code>	296
9.36.2.7	<code>fasp_blas_dcsr_m xv_agg()</code>	297
9.36.2.8	<code>fasp_blas_dcsr_ptap()</code>	297
9.36.2.9	<code>fasp_blas_dcsr_rap()</code>	298
9.36.2.10	<code>fasp_blas_dcsr_rap2()</code>	299
9.36.2.11	<code>fasp_blas_dcsr_rap4()</code>	300
9.36.2.12	<code>fasp_blas_dcsr_rap_agg()</code>	300
9.36.2.13	<code>fasp_blas_dcsr_rap_agg1()</code>	301
9.36.2.14	<code>fasp_blas_dcsr_vmv()</code>	302
9.37	BlaSpmvCSRL.c File Reference	302
9.37.1	Detailed Description	303
9.37.2	Function Documentation	303



9.37.2.1 fasp_blas_dcsr_l_mvx()	303
9.38 BlaSpmvSTR.c File Reference	303
9.38.1 Detailed Description	304
9.38.2 Function Documentation	304
9.38.2.1 fasp_blas_dstr_aApy()	304
9.38.2.2 fasp_blas_dstr_mvx()	305
9.38.2.3 fasp_dstr_diagscale()	305
9.39 BlaVector.c File Reference	306
9.39.1 Detailed Description	307
9.39.2 Function Documentation	307
9.39.2.1 fasp_blas_dvec_axpy()	307
9.39.2.2 fasp_blas_dvec_axpyz()	307
9.39.2.3 fasp_blas_dvec_dotprod()	308
9.39.2.4 fasp_blas_dvec_norm1()	309
9.39.2.5 fasp_blas_dvec_norm2()	309
9.39.2.6 fasp_blas_dvec_norminf()	310
9.39.2.7 fasp_blas_dvec_relerr()	311
9.40 doxygen.h File Reference	312
9.40.1 Detailed Description	312
9.41 fasp.h File Reference	312
9.41.1 Detailed Description	315
9.41.2 Macro Definition Documentation	315
9.41.2.1 __FASP_HEADER__	315
9.41.2.2 ABS	315
9.41.2.3 DIAGONAL_PREF	316
9.41.2.4 DLMALLOC	316
9.41.2.5 FASP_GSRB	316
9.41.2.6 FASP_VERSION	316

9.41.2.7	GE	317
9.41.2.8	GT	317
9.41.2.9	INT	317
9.41.2.10	ISNAN	317
9.41.2.11	LE	318
9.41.2.12	LONG	318
9.41.2.13	LONGLONG	318
9.41.2.14	LS	318
9.41.2.15	MAX	319
9.41.2.16	MIN	319
9.41.2.17	NEDMALLOC	319
9.41.2.18	PUT_INT	319
9.41.2.19	PUT_REAL	320
9.41.2.20	REAL	320
9.41.2.21	RS_C1	320
9.41.2.22	SHORT	320
9.41.3	Typedef Documentation	321
9.41.3.1	dCOOmat	321
9.41.3.2	dCSRLmat	321
9.41.3.3	dCSRmat	321
9.41.3.4	ddenmat	321
9.41.3.5	dSTRmat	321
9.41.3.6	dvector	321
9.41.3.7	iCOOmat	322
9.41.3.8	iCSRmat	322
9.41.3.9	idenmat	322
9.41.3.10	ivector	322
9.41.4	Variable Documentation	322

9.41.4.1	count	322
9.41.4.2	IMAP	322
9.41.4.3	MAXIMAP	323
9.41.4.4	nx_rb	323
9.41.4.5	ny_rb	323
9.41.4.6	nz_rb	323
9.41.4.7	total_alloc_count	323
9.41.4.8	total_alloc_mem	323
9.42	fasp_block.h File Reference	324
9.42.1	Detailed Description	325
9.42.2	Macro Definition Documentation	325
9.42.2.1	__FASPBLOCK_HEADER__	325
9.42.3	Typedef Documentation	325
9.42.3.1	block_dvector	325
9.42.3.2	block_ivector	325
9.42.3.3	dBLCmat	326
9.42.3.4	dBSRmat	326
9.42.3.5	iBLCmat	326
9.43	fasp_const.h File Reference	326
9.43.1	Detailed Description	330
9.43.2	Macro Definition Documentation	330
9.43.2.1	AML_CYCLE	330
9.43.2.2	ASCEND	330
9.43.2.3	BIGREAL	330
9.43.2.4	CF_ORDER	331
9.43.2.5	CGPT	331
9.43.2.6	CLASSIC_AMG	331
9.43.2.7	COARSE_AC	331

9.43.2.8	COARSE_CR	331
9.43.2.9	COARSE_MIS	332
9.43.2.10	COARSE_RS	332
9.43.2.11	COARSE_RSP	332
9.43.2.12	CPFIRST	332
9.43.2.13	DESCEND	332
9.43.2.14	ERROR_ALLOC_MEM	333
9.43.2.15	ERROR_AMG_COARSE_TYPE	333
9.43.2.16	ERROR_AMG_COARSEING	333
9.43.2.17	ERROR_AMG_INTERP_TYPE	333
9.43.2.18	ERROR_AMG_SMOOTH_TYPE	333
9.43.2.19	ERROR_DATA_STRUCTURE	334
9.43.2.20	ERROR_DATA_ZERODIAG	334
9.43.2.21	ERROR_DUMMY_VAR	334
9.43.2.22	ERROR_INPUT_PAR	334
9.43.2.23	ERROR_LIC_TYPE	334
9.43.2.24	ERROR_MAT_SIZE	335
9.43.2.25	ERROR_MISC	335
9.43.2.26	ERROR_NUM_BLOCKS	335
9.43.2.27	ERROR_OPEN_FILE	335
9.43.2.28	ERROR_QUAD_DIM	335
9.43.2.29	ERROR_QUAD_TYPE	336
9.43.2.30	ERROR_REGRESS	336
9.43.2.31	ERROR_SOLVER_EXIT	336
9.43.2.32	ERROR_SOLVER_ILUSETUP	336
9.43.2.33	ERROR_SOLVER_MAXIT	336
9.43.2.34	ERROR_SOLVER_MISC	337
9.43.2.35	ERROR_SOLVER_PRECTYPE	337

9.43.2.36 ERROR_SOLVER_SOLSTAG . . . . .	337
9.43.2.37 ERROR_SOLVER_STAG . . . . .	337
9.43.2.38 ERROR_SOLVER_TOLSMALL . . . . .	337
9.43.2.39 ERROR_SOLVER_TYPE . . . . .	338
9.43.2.40 ERROR_UNKNOWN . . . . .	338
9.43.2.41 ERROR_WRONG_FILE . . . . .	338
9.43.2.42 FALSE . . . . .	338
9.43.2.43 FASP_SUCCESS . . . . .	338
9.43.2.44 FGPT . . . . .	339
9.43.2.45 FPFIRST . . . . .	339
9.43.2.46 G0PT . . . . .	339
9.43.2.47 ILU_MC_OMP . . . . .	339
9.43.2.48 ILUk . . . . .	340
9.43.2.49 ILU <sub>t</sub> . . . . .	340
9.43.2.50 ILU <sub>tp</sub> . . . . .	340
9.43.2.51 INTERP_DIR . . . . .	340
9.43.2.52 INTERP_ENG . . . . .	341
9.43.2.53 INTERP_EXT . . . . .	341
9.43.2.54 INTERP_STD . . . . .	341
9.43.2.55 ISPT . . . . .	341
9.43.2.56 MAT_bBSR . . . . .	341
9.43.2.57 MAT_bCSR . . . . .	342
9.43.2.58 MAT_BLC . . . . .	342
9.43.2.59 MAT_BSR . . . . .	342
9.43.2.60 MAT_bSTR . . . . .	342
9.43.2.61 MAT_CSR . . . . .	342
9.43.2.62 MAT_CSRL . . . . .	343
9.43.2.63 MAT_FREE . . . . .	343

9.43.2.64 MAT_STR . . . . .	343
9.43.2.65 MAT_SymCSR . . . . .	343
9.43.2.66 MAX_AMG_LVL . . . . .	343
9.43.2.67 MAX_CRATE . . . . .	344
9.43.2.68 MAX_REFINE_LVL . . . . .	344
9.43.2.69 MAX_RESTART . . . . .	344
9.43.2.70 MAX_STAG . . . . .	344
9.43.2.71 MIN_CDOF . . . . .	344
9.43.2.72 MIN_CRATE . . . . .	345
9.43.2.73 NL_AMLI_CYCLE . . . . .	345
9.43.2.74 NO_ORDER . . . . .	345
9.43.2.75 OFF . . . . .	345
9.43.2.76 ON . . . . .	346
9.43.2.77 OPENMP_HOLDS . . . . .	346
9.43.2.78 PAIRWISE . . . . .	346
9.43.2.79 PREC_AMG . . . . .	346
9.43.2.80 PREC_DIAG . . . . .	347
9.43.2.81 PREC_FMG . . . . .	347
9.43.2.82 PREC_ILU . . . . .	347
9.43.2.83 PREC_NULL . . . . .	347
9.43.2.84 PREC_SCHWARZ . . . . .	347
9.43.2.85 PRINT_ALL . . . . .	348
9.43.2.86 PRINT_MIN . . . . .	348
9.43.2.87 PRINT_MORE . . . . .	348
9.43.2.88 PRINT_MOST . . . . .	348
9.43.2.89 PRINT_NONE . . . . .	348
9.43.2.90 PRINT_SOME . . . . .	349
9.43.2.91 SA_AMG . . . . .	349

9.43.2.92 SCHWARZ_BACKWARD . . . . .	349
9.43.2.93 SCHWARZ_FORWARD . . . . .	349
9.43.2.94 SCHWARZ_SYMMETRIC . . . . .	349
9.43.2.95 SMALLREAL . . . . .	350
9.43.2.96 SMALLREAL2 . . . . .	350
9.43.2.97 SMOOTHER_BLKOIL . . . . .	350
9.43.2.98 SMOOTHER_CG . . . . .	350
9.43.2.99 SMOOTHER_GS . . . . .	350
9.43.2.100 SMOOTHER_GSOR . . . . .	351
9.43.2.101 SMOOTHER_JACOBI . . . . .	351
9.43.2.102 SMOOTHER_L1DIAG . . . . .	351
9.43.2.103 SMOOTHER_POLY . . . . .	351
9.43.2.104 SMOOTHER_SGS . . . . .	351
9.43.2.105 SMOOTHER_SGSOR . . . . .	352
9.43.2.106 SMOOTHER_SOR . . . . .	352
9.43.2.107 SMOOTHER_SPETEN . . . . .	352
9.43.2.108 SMOOTHER_SSOR . . . . .	352
9.43.2.109 SOLVER_AMG . . . . .	352
9.43.2.110 SOLVER_BiCGstab . . . . .	353
9.43.2.111 SOLVER_CG . . . . .	353
9.43.2.112 SOLVER_DEFAULT . . . . .	353
9.43.2.113 SOLVER_FMG . . . . .	353
9.43.2.114 SOLVER_GCG . . . . .	353
9.43.2.115 SOLVER_GCR . . . . .	354
9.43.2.116 SOLVER_GMRES . . . . .	354
9.43.2.117 SOLVER_MinRes . . . . .	354
9.43.2.118 SOLVER_MUMPS . . . . .	354
9.43.2.119 SOLVER_PARDISO . . . . .	354

9.43.2.120	SOLVER_SBiCGstab	355
9.43.2.121	SOLVER_SCG	355
9.43.2.122	SOLVER_SGCG	355
9.43.2.123	SOLVER_SGMRES	355
9.43.2.124	SOLVER_SMinRes	355
9.43.2.125	SOLVER_SUPERLU	356
9.43.2.126	SOLVER_SVFGMRES	356
9.43.2.127	SOLVER_SVGMRES	356
9.43.2.128	SOLVER_UMFPACK	356
9.43.2.129	SOLVER_VBiCGstab	356
9.43.2.130	SOLVER_VFGMRES	357
9.43.2.131	SOLVER_VGMRES	357
9.43.2.132	STAG_RATIO	357
9.43.2.133	STOP_MOD_REL_RES	357
9.43.2.134	STOP_REL_PRECRES	357
9.43.2.135	STOP_REL_RES	358
9.43.2.136	TRUE	358
9.43.2.137	UA_AMG	358
9.43.2.138	UNPT	358
9.43.2.139	USERDEFINED	359
9.43.2.140	V_CYCLE	359
9.43.2.141	VMB	359
9.43.2.142	W_CYCLE	359
9.44	fasp_grid.h File Reference	359
9.44.1	Detailed Description	360
9.44.2	Macro Definition Documentation	360
9.44.2.1	__FASPGRID_HEADER__	360
9.44.3	Typedef Documentation	360



9.44.3.1	grid2d	360
9.44.3.2	pcgrid2d	361
9.44.3.3	pgrid2d	361
9.45	ItrSmootherBSR.c File Reference	361
9.45.1	Detailed Description	362
9.45.2	Function Documentation	362
9.45.2.1	fasp_smoother_dbsr_gs()	362
9.45.2.2	fasp_smoother_dbsr_gs1()	363
9.45.2.3	fasp_smoother_dbsr_gs_ascend()	364
9.45.2.4	fasp_smoother_dbsr_gs_ascend1()	364
9.45.2.5	fasp_smoother_dbsr_gs_descend()	365
9.45.2.6	fasp_smoother_dbsr_gs_descend1()	366
9.45.2.7	fasp_smoother_dbsr_gs_order1()	366
9.45.2.8	fasp_smoother_dbsr_gs_order2()	367
9.45.2.9	fasp_smoother_dbsr_ilu()	368
9.45.2.10	fasp_smoother_dbsr_jacobi()	368
9.45.2.11	fasp_smoother_dbsr_jacobi1()	369
9.45.2.12	fasp_smoother_dbsr_jacobi_setup()	370
9.45.2.13	fasp_smoother_dbsr_sor()	370
9.45.2.14	fasp_smoother_dbsr_sor1()	371
9.45.2.15	fasp_smoother_dbsr_sor_ascend()	372
9.45.2.16	fasp_smoother_dbsr_sor_descend()	372
9.45.2.17	fasp_smoother_dbsr_sor_order()	373
9.45.3	Variable Documentation	374
9.45.3.1	ilu_solve_omp	374
9.46	ItrSmootherCSR.c File Reference	374
9.46.1	Detailed Description	375
9.46.2	Function Documentation	375

9.46.2.1	<a href="#">fasp_smoother_dcsr_gs()</a>	375
9.46.2.2	<a href="#">fasp_smoother_dcsr_gs_cf()</a>	376
9.46.2.3	<a href="#">fasp_smoother_dcsr_gs_rb3d()</a>	377
9.46.2.4	<a href="#">fasp_smoother_dcsr_ilu()</a>	377
9.46.2.5	<a href="#">fasp_smoother_dcsr_jacobi()</a>	378
9.46.2.6	<a href="#">fasp_smoother_dcsr_kaczmarz()</a>	379
9.46.2.7	<a href="#">fasp_smoother_dcsr_L1diag()</a>	380
9.46.2.8	<a href="#">fasp_smoother_dcsr_sgs()</a>	381
9.46.2.9	<a href="#">fasp_smoother_dcsr_sor()</a>	381
9.46.2.10	<a href="#">fasp_smoother_dcsr_sor_cf()</a>	382
9.47	<a href="#">ltrSmootherCSRcr.c File Reference</a>	383
9.47.1	<a href="#">Detailed Description</a>	383
9.47.2	<a href="#">Function Documentation</a>	383
9.47.2.1	<a href="#">fasp_smoother_dcsr_gscr()</a>	383
9.48	<a href="#">ltrSmootherCSRpoly.c File Reference</a>	384
9.48.1	<a href="#">Detailed Description</a>	385
9.48.2	<a href="#">Function Documentation</a>	385
9.48.2.1	<a href="#">fasp_smoother_dcsr_poly()</a>	385
9.48.2.2	<a href="#">fasp_smoother_dcsr_poly_old()</a>	386
9.49	<a href="#">ltrSmootherSTR.c File Reference</a>	386
9.49.1	<a href="#">Detailed Description</a>	387
9.49.2	<a href="#">Function Documentation</a>	388
9.49.2.1	<a href="#">fasp_generate_diaginv_block()</a>	388
9.49.2.2	<a href="#">fasp_smoother_dstr_gs()</a>	388
9.49.2.3	<a href="#">fasp_smoother_dstr_gs1()</a>	389
9.49.2.4	<a href="#">fasp_smoother_dstr_gs_ascend()</a>	390
9.49.2.5	<a href="#">fasp_smoother_dstr_gs_cf()</a>	390
9.49.2.6	<a href="#">fasp_smoother_dstr_gs_descend()</a>	392

9.49.2.7	<code>fasp_smoother_dstr_gs_order()</code>	393
9.49.2.8	<code>fasp_smoother_dstr_jacobi()</code>	393
9.49.2.9	<code>fasp_smoother_dstr_jacobi1()</code>	394
9.49.2.10	<code>fasp_smoother_dstr_schwarz()</code>	395
9.49.2.11	<code>fasp_smoother_dstr_sor()</code>	395
9.49.2.12	<code>fasp_smoother_dstr_sor1()</code>	396
9.49.2.13	<code>fasp_smoother_dstr_sor_ascend()</code>	397
9.49.2.14	<code>fasp_smoother_dstr_sor_cf()</code>	398
9.49.2.15	<code>fasp_smoother_dstr_sor_descend()</code>	398
9.49.2.16	<code>fasp_smoother_dstr_sor_order()</code>	399
9.50	KryPbcgs.c File Reference	400
9.50.1	Detailed Description	400
9.50.2	Function Documentation	401
9.50.2.1	<code>fasp_solver_dblc_pbcgs()</code>	401
9.50.2.2	<code>fasp_solver_dbsr_pbcgs()</code>	402
9.50.2.3	<code>fasp_solver_dcsr_pbcgs()</code>	403
9.50.2.4	<code>fasp_solver_dstr_pbcgs()</code>	404
9.50.2.5	<code>fasp_solver_pbcgs()</code>	405
9.51	KryPcg.c File Reference	405
9.51.1	Detailed Description	406
9.51.2	Function Documentation	407
9.51.2.1	<code>fasp_solver_dblc_pcg()</code>	407
9.51.2.2	<code>fasp_solver_dbsr_pcg()</code>	408
9.51.2.3	<code>fasp_solver_dcsr_pcg()</code>	409
9.51.2.4	<code>fasp_solver_dstr_pcg()</code>	410
9.51.2.5	<code>fasp_solver_pcg()</code>	411
9.52	KryPcg.c File Reference	412
9.52.1	Detailed Description	412

9.52.2	Function Documentation	412
9.52.2.1	<code>fasp_solver_dcsr_pgcg()</code>	412
9.52.2.2	<code>fasp_solver_pgcg()</code>	413
9.53	KryPgcr.c File Reference	414
9.53.1	Detailed Description	414
9.53.2	Function Documentation	415
9.53.2.1	<code>fasp_solver_dcsr_pgcr()</code>	415
9.54	KryPgmres.c File Reference	416
9.54.1	Detailed Description	416
9.54.2	Function Documentation	416
9.54.2.1	<code>fasp_solver_dblc_pgmres()</code>	417
9.54.2.2	<code>fasp_solver_dbsr_pgmres()</code>	418
9.54.2.3	<code>fasp_solver_dcsr_pgmres()</code>	419
9.54.2.4	<code>fasp_solver_dstr_pgmres()</code>	420
9.54.2.5	<code>fasp_solver_pgmres()</code>	421
9.55	KryPminres.c File Reference	422
9.55.1	Detailed Description	422
9.55.2	Function Documentation	422
9.55.2.1	<code>fasp_solver_dblc_pminres()</code>	422
9.55.2.2	<code>fasp_solver_dcsr_pminres()</code>	423
9.55.2.3	<code>fasp_solver_dstr_pminres()</code>	424
9.55.2.4	<code>fasp_solver_pminres()</code>	425
9.56	KryPvbcgs.c File Reference	426
9.56.1	Detailed Description	427
9.56.2	Function Documentation	427
9.56.2.1	<code>fasp_solver_dblc_pvbcgs()</code>	427
9.56.2.2	<code>fasp_solver_dbsr_pvbcgs()</code>	428
9.56.2.3	<code>fasp_solver_dcsr_pvbcgs()</code>	429

9.56.2.4	<a href="#">fasp_solver_dstr_pvbcgs()</a>	430
9.56.2.5	<a href="#">fasp_solver_pvbcgs()</a>	430
9.57	<a href="#">KryPvfgmres.c File Reference</a>	432
9.57.1	<a href="#">Detailed Description</a>	433
9.57.2	<a href="#">Function Documentation</a>	433
9.57.2.1	<a href="#">fasp_solver_dblc_pvfgmres()</a>	433
9.57.2.2	<a href="#">fasp_solver_dbsr_pvfgmres()</a>	434
9.57.2.3	<a href="#">fasp_solver_dcsr_pvfgmres()</a>	435
9.57.2.4	<a href="#">fasp_solver_pvfgmres()</a>	436
9.58	<a href="#">KryPvgmres.c File Reference</a>	437
9.58.1	<a href="#">Detailed Description</a>	438
9.58.2	<a href="#">Function Documentation</a>	438
9.58.2.1	<a href="#">fasp_solver_dblc_pvgmres()</a>	438
9.58.2.2	<a href="#">fasp_solver_dbsr_pvgmres()</a>	439
9.58.2.3	<a href="#">fasp_solver_dcsr_pvgmres()</a>	440
9.58.2.4	<a href="#">fasp_solver_dstr_pvgmres()</a>	441
9.58.2.5	<a href="#">fasp_solver_pvgmres()</a>	442
9.59	<a href="#">KrySPbcgs.c File Reference</a>	443
9.59.1	<a href="#">Detailed Description</a>	443
9.59.2	<a href="#">Function Documentation</a>	444
9.59.2.1	<a href="#">fasp_solver_dblc_spbcgs()</a>	444
9.59.2.2	<a href="#">fasp_solver_dbsr_spbcgs()</a>	445
9.59.2.3	<a href="#">fasp_solver_dcsr_spbcgs()</a>	445
9.59.2.4	<a href="#">fasp_solver_dstr_spbcgs()</a>	447
9.60	<a href="#">KrySPcg.c File Reference</a>	448
9.60.1	<a href="#">Detailed Description</a>	448
9.60.2	<a href="#">Function Documentation</a>	449
9.60.2.1	<a href="#">fasp_solver_dblc_spcg()</a>	449

9.60.2.2	<code>fasp_solver_dcsr_spcg()</code>	450
9.60.2.3	<code>fasp_solver_dstr_spcg()</code>	450
9.61	KrySPgmres.c File Reference	452
9.61.1	Detailed Description	453
9.61.2	Function Documentation	453
9.61.2.1	<code>fasp_solver_dblc_spgmres()</code>	453
9.61.2.2	<code>fasp_solver_dbsr_spgmres()</code>	454
9.61.2.3	<code>fasp_solver_dcsr_spgmres()</code>	455
9.61.2.4	<code>fasp_solver_dstr_spgmres()</code>	456
9.62	KrySPminres.c File Reference	456
9.62.1	Detailed Description	457
9.62.2	Function Documentation	457
9.62.2.1	<code>fasp_solver_dblc_spminres()</code>	457
9.62.2.2	<code>fasp_solver_dcsr_spminres()</code>	458
9.62.2.3	<code>fasp_solver_dstr_spminres()</code>	459
9.63	KrySPvgmres.c File Reference	460
9.63.1	Detailed Description	460
9.63.2	Function Documentation	460
9.63.2.1	<code>fasp_solver_dblc_spvgmres()</code>	461
9.63.2.2	<code>fasp_solver_dbsr_spvgmres()</code>	462
9.63.2.3	<code>fasp_solver_dcsr_spvgmres()</code>	463
9.63.2.4	<code>fasp_solver_dstr_spvgmres()</code>	464
9.64	PreAMGCoarsenCR.c File Reference	464
9.64.1	Detailed Description	465
9.64.2	Function Documentation	465
9.64.2.1	<code>fasp_amg_coarsening_cr()</code>	465
9.65	PreAMGCoarsenRS.c File Reference	466
9.65.1	Detailed Description	466

9.65.2	Function Documentation	466
9.65.2.1	fasp_amg_coarsening_rs()	467
9.66	PreAMGInterp.c File Reference	467
9.66.1	Detailed Description	468
9.66.2	Function Documentation	468
9.66.2.1	fasp_amg_interp()	468
9.66.2.2	fasp_amg_interp_trunc()	469
9.67	PreAMGInterpEM.c File Reference	469
9.67.1	Detailed Description	470
9.67.2	Function Documentation	470
9.67.2.1	fasp_amg_interp_em()	470
9.68	PreAMGSetupCR.c File Reference	471
9.68.1	Detailed Description	471
9.68.2	Function Documentation	471
9.68.2.1	fasp_amg_setup_cr()	471
9.69	PreAMGSetupRS.c File Reference	472
9.69.1	Detailed Description	472
9.69.2	Function Documentation	473
9.69.2.1	fasp_amg_setup_rs()	473
9.70	PreAMGSetupSA.c File Reference	473
9.70.1	Detailed Description	474
9.70.2	Function Documentation	474
9.70.2.1	fasp_amg_setup_sa()	474
9.71	PreAMGSetupSABSR.c File Reference	475
9.71.1	Detailed Description	475
9.71.2	Function Documentation	475
9.71.2.1	fasp_amg_setup_sa_bsr()	475
9.72	PreAMGSetupUA.c File Reference	476

9.72.1 Detailed Description . . . . .	476
9.72.2 Function Documentation . . . . .	477
9.72.2.1 fasp_amg_setup_ua() . . . . .	477
9.73 PreAMGSetupUABSR.c File Reference . . . . .	477
9.73.1 Detailed Description . . . . .	478
9.73.2 Function Documentation . . . . .	478
9.73.2.1 fasp_amg_setup_ua_bsr() . . . . .	478
9.74 PreBLC.c File Reference . . . . .	479
9.74.1 Detailed Description . . . . .	479
9.74.2 Function Documentation . . . . .	480
9.74.2.1 fasp_precond_block_diag_3() . . . . .	480
9.74.2.2 fasp_precond_block_diag_3_amg() . . . . .	480
9.74.2.3 fasp_precond_block_diag_4() . . . . .	481
9.74.2.4 fasp_precond_block_lower_3() . . . . .	481
9.74.2.5 fasp_precond_block_lower_3_amg() . . . . .	482
9.74.2.6 fasp_precond_block_lower_4() . . . . .	483
9.74.2.7 fasp_precond_block_SGS_3() . . . . .	483
9.74.2.8 fasp_precond_block_SGS_3_amg() . . . . .	484
9.74.2.9 fasp_precond_block_upper_3() . . . . .	484
9.74.2.10 fasp_precond_block_upper_3_amg() . . . . .	485
9.74.2.11 fasp_precond_sweeping() . . . . .	485
9.75 PreBSR.c File Reference . . . . .	486
9.75.1 Detailed Description . . . . .	487
9.75.2 Function Documentation . . . . .	487
9.75.2.1 fasp_precond_dbsr_amg() . . . . .	487
9.75.2.2 fasp_precond_dbsr_amg_nk() . . . . .	487
9.75.2.3 fasp_precond_dbsr_diag() . . . . .	488
9.75.2.4 fasp_precond_dbsr_diag_nc2() . . . . .	489



9.75.2.5	<a href="#">fasp_precond_dbsr_diag_nc3()</a>	489
9.75.2.6	<a href="#">fasp_precond_dbsr_diag_nc5()</a>	490
9.75.2.7	<a href="#">fasp_precond_dbsr_diag_nc7()</a>	491
9.75.2.8	<a href="#">fasp_precond_dbsr_ilu()</a>	492
9.75.2.9	<a href="#">fasp_precond_dbsr_ilu_ls_omp()</a>	492
9.75.2.10	<a href="#">fasp_precond_dbsr_ilu_mc_omp()</a>	493
9.75.2.11	<a href="#">fasp_precond_dbsr_nl_amli()</a>	494
9.76	<a href="#">PreCSR.c File Reference</a>	494
9.76.1	<a href="#">Detailed Description</a>	495
9.76.2	<a href="#">Function Documentation</a>	495
9.76.2.1	<a href="#">fasp_precond_amg()</a>	495
9.76.2.2	<a href="#">fasp_precond_amg_nk()</a>	496
9.76.2.3	<a href="#">fasp_precond_amli()</a>	497
9.76.2.4	<a href="#">fasp_precond_diag()</a>	497
9.76.2.5	<a href="#">fasp_precond_famg()</a>	498
9.76.2.6	<a href="#">fasp_precond_free()</a>	498
9.76.2.7	<a href="#">fasp_precond_ilu()</a>	499
9.76.2.8	<a href="#">fasp_precond_ilu_backward()</a>	500
9.76.2.9	<a href="#">fasp_precond_ilu_forward()</a>	501
9.76.2.10	<a href="#">fasp_precond_nl_amli()</a>	502
9.76.2.11	<a href="#">fasp_precond_schwarz()</a>	502
9.76.2.12	<a href="#">fasp_precond_setup()</a>	503
9.77	<a href="#">PreDataInit.c File Reference</a>	503
9.77.1	<a href="#">Detailed Description</a>	504
9.77.2	<a href="#">Function Documentation</a>	504
9.77.2.1	<a href="#">fasp_amg_data_bsr_create()</a>	504
9.77.2.2	<a href="#">fasp_amg_data_bsr_free()</a>	505
9.77.2.3	<a href="#">fasp_amg_data_create()</a>	505

9.77.2.4	<code>fasp_amg_data_free()</code>	506
9.77.2.5	<code>fasp_ilu_data_create()</code>	507
9.77.2.6	<code>fasp_ilu_data_free()</code>	507
9.77.2.7	<code>fasp_ilu_data_null()</code>	508
9.77.2.8	<code>fasp_precond_data_null()</code>	508
9.77.2.9	<code>fasp_precond_null()</code>	509
9.77.2.10	<code>fasp_schwarz_data_free()</code>	509
9.78	PreMGCycle.c File Reference	510
9.78.1	Detailed Description	510
9.78.2	Function Documentation	510
9.78.2.1	<code>fasp_solver_mgcycle()</code>	510
9.78.2.2	<code>fasp_solver_mgcycle_bsr()</code>	511
9.79	PreMGCycleFull.c File Reference	511
9.79.1	Detailed Description	512
9.79.2	Function Documentation	512
9.79.2.1	<code>fasp_solver_fmecycle()</code>	512
9.80	PreMGRecur.c File Reference	513
9.80.1	Detailed Description	513
9.80.2	Function Documentation	513
9.80.2.1	<code>fasp_solver_mgrecur()</code>	513
9.81	PreMGRecurAMLI.c File Reference	514
9.81.1	Detailed Description	515
9.81.2	Function Documentation	515
9.81.2.1	<code>fasp_amg_amli_coef()</code>	515
9.81.2.2	<code>fasp_solver_amli()</code>	516
9.81.2.3	<code>fasp_solver_nl_amli()</code>	516
9.81.2.4	<code>fasp_solver_nl_amli_bsr()</code>	517
9.82	PreMGsolve.c File Reference	518

9.82.1 Detailed Description	518
9.82.2 Function Documentation	519
9.82.2.1 fasp_amg_solve()	519
9.82.2.2 fasp_amg_solve_amli()	519
9.82.2.3 fasp_amg_solve_nl_amli()	520
9.82.2.4 fasp_famg_solve()	521
9.83 PreSTR.c File Reference	521
9.83.1 Detailed Description	522
9.83.2 Function Documentation	522
9.83.2.1 fasp_precond_dstr_blockgs()	522
9.83.2.2 fasp_precond_dstr_diag()	523
9.83.2.3 fasp_precond_dstr_ilu0()	523
9.83.2.4 fasp_precond_dstr_ilu0_backward()	524
9.83.2.5 fasp_precond_dstr_ilu0_forward()	525
9.83.2.6 fasp_precond_dstr_ilu1()	525
9.83.2.7 fasp_precond_dstr_ilu1_backward()	526
9.83.2.8 fasp_precond_dstr_ilu1_forward()	526
9.84 SolAMG.c File Reference	527
9.84.1 Detailed Description	527
9.84.2 Function Documentation	527
9.84.2.1 fasp_solver_amg()	527
9.85 SolBLC.c File Reference	528
9.85.1 Detailed Description	529
9.85.2 Function Documentation	529
9.85.2.1 fasp_solver_dbldc_itsolver()	529
9.85.2.2 fasp_solver_dbldc_krylov()	530
9.85.2.3 fasp_solver_dbldc_krylov_block_3()	530
9.85.2.4 fasp_solver_dbldc_krylov_block_4()	531

9.85.2.5	<code>fasp_solver_dblc_krylov_sweeping()</code>	532
9.86	SolBSR.c File Reference	533
9.86.1	Detailed Description	534
9.86.2	Function Documentation	534
9.86.2.1	<code>fasp_solver_dbsr_itsolver()</code>	534
9.86.2.2	<code>fasp_solver_dbsr_krylov()</code>	535
9.86.2.3	<code>fasp_solver_dbsr_krylov_amg()</code>	535
9.86.2.4	<code>fasp_solver_dbsr_krylov_amg_nk()</code>	536
9.86.2.5	<code>fasp_solver_dbsr_krylov_diag()</code>	537
9.86.2.6	<code>fasp_solver_dbsr_krylov_ilu()</code>	538
9.86.2.7	<code>fasp_solver_dbsr_krylov_nk_amg()</code>	538
9.87	SolCSR.c File Reference	539
9.87.1	Detailed Description	540
9.87.2	Function Documentation	540
9.87.2.1	<code>fasp_solver_dcsr_itsolver()</code>	540
9.87.2.2	<code>fasp_solver_dcsr_krylov()</code>	541
9.87.2.3	<code>fasp_solver_dcsr_krylov_amg()</code>	542
9.87.2.4	<code>fasp_solver_dcsr_krylov_amg_nk()</code>	543
9.87.2.5	<code>fasp_solver_dcsr_krylov_diag()</code>	543
9.87.2.6	<code>fasp_solver_dcsr_krylov_ilu()</code>	544
9.87.2.7	<code>fasp_solver_dcsr_krylov_ilu_M()</code>	545
9.87.2.8	<code>fasp_solver_dcsr_krylov_schwarz()</code>	546
9.88	SolFAMG.c File Reference	546
9.88.1	Detailed Description	547
9.88.2	Function Documentation	547
9.88.2.1	<code>fasp_solver_famg()</code>	547
9.89	SolGMGPoisson.c File Reference	548
9.89.1	Detailed Description	548

9.89.2	Function Documentation	549
9.89.2.1	fasp_poisson_fgm1d()	549
9.89.2.2	fasp_poisson_fgm2d()	549
9.89.2.3	fasp_poisson_fgm3d()	550
9.89.2.4	fasp_poisson_gm1d()	551
9.89.2.5	fasp_poisson_gm2d()	552
9.89.2.6	fasp_poisson_gm3d()	552
9.89.2.7	fasp_poisson_gmgcg1d()	553
9.89.2.8	fasp_poisson_gmgcg2d()	554
9.89.2.9	fasp_poisson_gmgcg3d()	555
9.90	SolMatFree.c File Reference	556
9.90.1	Detailed Description	556
9.90.2	Function Documentation	556
9.90.2.1	fasp_solver_itsolver()	556
9.90.2.2	fasp_solver_krylov()	557
9.90.2.3	fasp_solver_matfree_init()	558
9.91	SolSTR.c File Reference	559
9.91.1	Detailed Description	559
9.91.2	Function Documentation	559
9.91.2.1	fasp_solver_dstr_itsolver()	559
9.91.2.2	fasp_solver_dstr_krylov()	560
9.91.2.3	fasp_solver_dstr_krylov_blockgs()	561
9.91.2.4	fasp_solver_dstr_krylov_diag()	562
9.91.2.5	fasp_solver_dstr_krylov_ilu()	562
9.92	SolWrapper.c File Reference	563
9.92.1	Detailed Description	563
9.92.2	Function Documentation	564
9.92.2.1	fasp_fwrapper_amg_()	564

9.92.2.2	<a href="#">fasp_fwrapper_krylov_amg()</a>	565
9.92.2.3	<a href="#">fasp_wrapper_dbsr_krylov_amg()</a>	565
9.92.2.4	<a href="#">fasp_wrapper_dcoo_dbsr_krylov_amg()</a>	566
9.93	<a href="#">XtrMumps.c File Reference</a>	567
9.93.1	<a href="#">Detailed Description</a>	568
9.93.2	<a href="#">Macro Definition Documentation</a>	568
9.93.2.1	<a href="#">ICNTL</a>	568
9.93.3	<a href="#">Function Documentation</a>	568
9.93.3.1	<a href="#">fasp_solver_mumps()</a>	568
9.93.3.2	<a href="#">fasp_solver_mumps_steps()</a>	569
9.94	<a href="#">XtrPardiso.c File Reference</a>	570
9.94.1	<a href="#">Detailed Description</a>	570
9.94.2	<a href="#">Function Documentation</a>	570
9.94.2.1	<a href="#">fasp_solver_pardiso()</a>	570
9.95	<a href="#">XtrSamg.c File Reference</a>	571
9.95.1	<a href="#">Detailed Description</a>	571
9.95.2	<a href="#">Function Documentation</a>	571
9.95.2.1	<a href="#">dCSRmat2SAMGInput()</a>	571
9.95.2.2	<a href="#">dvector2SAMGInput()</a>	572
9.96	<a href="#">XtrSuperlu.c File Reference</a>	572
9.96.1	<a href="#">Detailed Description</a>	573
9.96.2	<a href="#">Function Documentation</a>	573
9.96.2.1	<a href="#">fasp_solver_superlu()</a>	573
9.97	<a href="#">XtrUmfpack.c File Reference</a>	574
9.97.1	<a href="#">Detailed Description</a>	574
9.97.2	<a href="#">Function Documentation</a>	574
9.97.2.1	<a href="#">fasp_solver_umfpack()</a>	574

# 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  $P \leftrightarrow DE$  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 structure of FASP is designed as follows: Level 0 (Aux\*.c): Auxiliary functions (timing, memory, ...) Level 1 (Bla\*.c): Basic linear algebra subroutines (SpMV, RAP, ILU, ...) Level 2 (Itr\*.c): Iterative methods and smoothers (Jacobi, GS, SOR, ...) Level 3 (Kry\*.c): Krylov iterative methods (CG, GMRES, ...) Level 4 (Pre\*.c): Preconditioners (GMG, AMG, ...) Level 5 (Sol\*.c): User interface for FASP solvers (Solvers, wrappers, ...) Level x (Xtr\*.c): Interface to external packages (Direct solvers, IO, ...)

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://fasp.sourceforge.net/download/faspsolver.zip>

We use HG (Mecurial) 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:

```
$ make config
```

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)
- Hu, Xiaozhe (Tufts University, USA)
- Li, Zheng (Kunming University of Science and Technology, China)
- Zhang, Chensong (Chinese Academy of Sciences, China)
- Zhang, Hongxuan (Penn State University, USA)

With contributions from (in alphabetic order):

- Brannick, James (Penn State University, USA)
- Chen, Long (University of California, Irvine, USA)
- Huang, Feiteng (Sichuan University, China)
- Huang, Xuehai (Shanghai Jiaotong University, 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, 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 solvers . . . . .	19
<a href="#">AMG_data_bsr</a>	Data for multigrid levels. (BSR format) . . . . .	20
<a href="#">AMG_param</a>	Parameters for AMG solver . . . . .	22
<a href="#">block_dvector</a>	Block REAL vector structure . . . . .	24
<a href="#">block_ivector</a>	Block INT vector structure . . . . .	25
<a href="#">dBLCmat</a>	Block REAL CSR matrix format . . . . .	26
<a href="#">dBSRmat</a>	Block sparse row storage matrix of REAL type . . . . .	26
<a href="#">dCOOmat</a>	Sparse matrix of REAL type in COO (or IJ) format . . . . .	27
<a href="#">dCSRLmat</a>	Sparse matrix of REAL type in CSRL format . . . . .	28
<a href="#">dCSRmat</a>	Sparse matrix of REAL type in CSR format . . . . .	29
<a href="#">ddenmat</a>	Dense matrix of REAL type . . . . .	30
<a href="#">dSTRmat</a>	Structure matrix of REAL type . . . . .	31
<a href="#">dvector</a>	Vector with n entries of REAL type . . . . .	32
<a href="#">grid2d</a>	Two dimensional grid data structure . . . . .	32
<a href="#">iBLCmat</a>	Block INT CSR matrix format . . . . .	35
<a href="#">iCOOmat</a>	Sparse matrix of INT type in COO (or IJ) format . . . . .	36

<a href="#">iCSRmat</a>	Sparse matrix of INT type in CSR format . . . . .	37
<a href="#">idenmat</a>	Dense matrix of INT type . . . . .	38
<a href="#">ILU_data</a>	Data for ILU setup . . . . .	38
<a href="#">ILU_param</a>	Parameters for ILU . . . . .	40
<a href="#">input_param</a>	Input parameters . . . . .	40
<a href="#">itsolver_param</a>	Parameters passed to iterative solvers . . . . .	52
<a href="#">ivector</a>	Vector with n entries of INT type . . . . .	54
<a href="#">mallinfo</a>	. . . . .	55
<a href="#">malloc_chunk</a>	. . . . .	56
<a href="#">malloc_params</a>	. . . . .	56
<a href="#">malloc_segment</a>	. . . . .	56
<a href="#">malloc_state</a>	. . . . .	57
<a href="#">malloc_tree_chunk</a>	. . . . .	58
<a href="#">Mumps_data</a>	Parameters for MUMPS interface . . . . .	58
<a href="#">mxv_matfree</a>	Matrix-vector multiplication, replace the actual matrix . . . . .	59
<a href="#">nedmallinfo</a>	. . . . .	59
<a href="#">Pardiso_data</a>	Parameters for Intel MKL PARDISO interface . . . . .	60
<a href="#">precond</a>	Preconditioner data and action . . . . .	60
<a href="#">precond_block_data</a>	Data passed to the preconditioner for block preconditioning for <a href="#">dBLCmat</a> format . . . . .	61
<a href="#">precond_data</a>	Data passed to the preconditioners . . . . .	62
<a href="#">precond_data_bsr</a>	Data passed to the preconditioners . . . . .	64
<a href="#">precond_data_str</a>	Data passed to the preconditioner for <a href="#">dSTRmat</a> matrices . . . . .	66
<a href="#">precond_diagbsr</a>	Data passed to diagonal preconditioner for <a href="#">dBSRmat</a> matrices . . . . .	67
<a href="#">precond_diagstr</a>	Data passed to diagonal preconditioner for <a href="#">dSTRmat</a> matrices . . . . .	68
<a href="#">precond_sweeping_data</a>	Data passed to the preconditioner for sweeping preconditioning . . . . .	69
<a href="#">Schwarz_data</a>	Data for Schwarz methods . . . . .	71
<a href="#">Schwarz_param</a>	Parameters for Schwarz method . . . . .	72

## 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 . . . . .	75
<a href="#">AuxConvert.c</a>	Some utilities for encoding format conversion . . . . .	81
<a href="#">AuxGivens.c</a>	Givens transformation . . . . .	83
<a href="#">AuxGraphics.c</a>	Graphical output for CSR matrix . . . . .	84
<a href="#">AuxInput.c</a>	Read and check input parameters . . . . .	88
<a href="#">AuxMemory.c</a>	Memory allocation and deallocation subroutines . . . . .	90
<a href="#">AuxMessage.c</a>	Output some useful messages . . . . .	95
<a href="#">AuxParam.c</a>	Initialize, set, or print input data and parameters . . . . .	99
<a href="#">AuxSort.c</a>	Subroutines for sorting, merging, removing duplicated integers . . . . .	111
<a href="#">AuxThreads.c</a>	Get and set number of threads and assign work load for each thread . . . . .	119
<a href="#">AuxTiming.c</a>	Timing subroutines . . . . .	122
<a href="#">AuxVector.c</a>	Simple operations for vectors . . . . .	123
<a href="#">BlaArray.c</a>	BLAS1 operations for arrays . . . . .	132
<a href="#">BlaEigen.c</a>	Subroutines for computing the extreme eigenvalues . . . . .	139
<a href="#">BlaFormat.c</a>	Subroutines for matrix format conversion . . . . .	140
<a href="#">BlaLU.c</a>	Incomplete LU decomposition: ILUk, ILUt, ILUtp . . . . .	147

<a href="#">BlaLUSetupBSR.c</a>	Setup incomplete LU decomposition for <a href="#">dBSRmat</a> matrices . . . . .	154
<a href="#">BlaLUSetupCSR.c</a>	Setup incomplete LU decomposition for <a href="#">dCSRmat</a> matrices . . . . .	158
<a href="#">BlaLUSetupSTR.c</a>	Setup incomplete LU decomposition for <a href="#">dSTRmat</a> matrices . . . . .	159
<a href="#">BlaIO.c</a>	Matrix/vector input/output subroutines . . . . .	161
<a href="#">BlaOrderingCSR.c</a>	Subroutines for generating ordering using algebraic information . . . . .	189
<a href="#">BlaSchwarzSetup.c</a>	Setup phase for the Schwarz methods . . . . .	190
<a href="#">BlaSmallMat.c</a>	BLAS operations for <i>small</i> dense matrices . . . . .	193
<a href="#">BlaSmallMatInv.c</a>	Find inversion of <i>small</i> dense matrices in row-major format . . . . .	215
<a href="#">BlaSmallMatLU.c</a>	LU decomposition and direct solver for small dense matrices . . . . .	224
<a href="#">BlaSparseBLC.c</a>	Sparse matrix block operations . . . . .	226
<a href="#">BlaSparseBSR.c</a>	Sparse matrix operations for <a href="#">dBSRmat</a> matrices . . . . .	228
<a href="#">BlaSparseCheck.c</a>	Check properties of sparse matrices . . . . .	240
<a href="#">BlaSparseCOO.c</a>	Sparse matrix operations for <a href="#">dCOOmat</a> matrices . . . . .	244
<a href="#">BlaSparseCSR.c</a>	Sparse matrix operations for <a href="#">dCSRmat</a> matrices . . . . .	247
<a href="#">BlaSparseCSRL.c</a>	Sparse matrix operations for <a href="#">dCSRLmat</a> matrices . . . . .	266
<a href="#">BlaSparseSTR.c</a>	Sparse matrix operations for <a href="#">dSTRmat</a> matrices . . . . .	267
<a href="#">BlaSparseUtil.c</a>	Routines for sparse matrix operations . . . . .	271
<a href="#">BlaSpmvBLC.c</a>	BLAS operations for <a href="#">dBLCmat</a> matrices . . . . .	282
<a href="#">BlaSpmvBSR.c</a>	BLAS operations for <a href="#">dBSRmat</a> matrices . . . . .	284
<a href="#">BlaSpmvCSR.c</a>	BLAS operations for <a href="#">dCSRmat</a> matrices . . . . .	291
<a href="#">BlaSpmvCSRL.c</a>	BLAS operations for <a href="#">dCSRLmat</a> matrices . . . . .	302
<a href="#">BlaSpmvSTR.c</a>	BLAS operations for <a href="#">dSTRmat</a> matrices . . . . .	303
<a href="#">BlaVector.c</a>	BLAS1 operations for vectors . . . . .	306
<a href="#">dlmalloc.h</a>	. . . . .	??
<a href="#">doxygen.h</a>	Main page for Doygen documentation . . . . .	312
<a href="#">fasp.h</a>	Main header file for FASP . . . . .	312
<a href="#">fasp_block.h</a>	Header file for FASP block matrices . . . . .	324

<a href="#">fasp_const.h</a>	Definition of all kinds of messages, including error messages, solver types, etc . . . . .	326
<a href="#">fasp_grid.h</a>	Header file for FASP grid . . . . .	359
<a href="#">hb_io.h</a>		??
<a href="#">ltrSmootherBSR.c</a>	Smoothers for <a href="#">dBSRmat</a> matrices . . . . .	361
<a href="#">ltrSmootherCSR.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices . . . . .	374
<a href="#">ltrSmootherCSRcr.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices using compatible relaxation . . . . .	383
<a href="#">ltrSmootherCSRpoly.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices using poly. approx. to $A^{-1}$ . . . . .	384
<a href="#">ltrSmootherSTR.c</a>	Smoothers for <a href="#">dSTRmat</a> matrices . . . . .	386
<a href="#">KryPbcgs.c</a>	Krylov subspace methods – Preconditioned BiCGstab . . . . .	400
<a href="#">KryPcg.c</a>	Krylov subspace methods – Preconditioned CG . . . . .	405
<a href="#">KryPgcg.c</a>	Krylov subspace methods – Preconditioned generalized CG . . . . .	412
<a href="#">KryPgcr.c</a>	Krylov subspace methods – Preconditioned GCR . . . . .	414
<a href="#">KryPgmmres.c</a>	Krylov subspace methods – Right-preconditioned GMRes . . . . .	416
<a href="#">KryPminres.c</a>	Krylov subspace methods – Preconditioned minimal residual . . . . .	422
<a href="#">KryPvbcgs.c</a>	Krylov subspace methods – Preconditioned BiCGstab . . . . .	426
<a href="#">KryPvfgmmres.c</a>	Krylov subspace methods – Preconditioned variable-restarting FGMMRes . . . . .	432
<a href="#">KryPvgmmres.c</a>	Krylov subspace methods – Preconditioned variable-restart GMRes . . . . .	437
<a href="#">KrySPbcgs.c</a>	Krylov subspace methods – Preconditioned BiCGstab with safety net . . . . .	443
<a href="#">KrySPcg.c</a>	Krylov subspace methods – Preconditioned CG with safety net . . . . .	448
<a href="#">KrySPgmmres.c</a>	Krylov subspace methods – Preconditioned GMRes with safety net . . . . .	452
<a href="#">KrySPminres.c</a>	Krylov subspace methods – Preconditioned minimal residual with safety net . . . . .	456
<a href="#">KrySPvgmmres.c</a>	Krylov subspace methods – Preconditioned variable-restart GMRes with safety net . . . . .	460
<a href="#">malloc.c.h</a>		??
<a href="#">nedmalloc.h</a>		??
<a href="#">PreAMGCoarsenCR.c</a>	Coarsening with Brannick-Falgout strategy . . . . .	464
<a href="#">PreAMGCoarsenRS.c</a>	Coarsening with a modified Ruge-Stuben strategy . . . . .	466
<a href="#">PreAMGInterp.c</a>	Direct and standard interpolations for classical AMG . . . . .	467
<a href="#">PreAMGInterpEM.c</a>	Interpolation operators for AMG based on energy-min . . . . .	469

<a href="#">PreAMGSetupCR.c</a>	
Brannick-Falgout compatible relaxation based AMG: SETUP phase	471
<a href="#">PreAMGSetupRS.c</a>	
Ruge-Stuben AMG: SETUP phase	472
<a href="#">PreAMGSetupSA.c</a>	
Smoothed aggregation AMG: SETUP phase	473
<a href="#">PreAMGSetupSABSR.c</a>	
Smoothed aggregation AMG: SETUP phase (for BSR matrices)	475
<a href="#">PreAMGSetupUA.c</a>	
Unsmoothed aggregation AMG: SETUP phase	476
<a href="#">PreAMGSetupUABSR.c</a>	
Unsmoothed aggregation AMG: SETUP phase (for BSR matrices)	477
<a href="#">PreBLC.c</a>	
Preconditioners for <a href="#">dBLCmat</a> matrices	479
<a href="#">PreBSR.c</a>	
Preconditioners for <a href="#">dBSRmat</a> matrices	486
<a href="#">PreCSR.c</a>	
Preconditioners for <a href="#">dCSRmat</a> matrices	494
<a href="#">PreDataInit.c</a>	
Initialize important data structures	503
<a href="#">PreMGCycle.c</a>	
Abstract multigrid cycle – non-recursive version	510
<a href="#">PreMGCycleFull.c</a>	
Abstract non-recursive full multigrid cycle	511
<a href="#">PreMGRecur.c</a>	
Abstract multigrid cycle – recursive version	513
<a href="#">PreMGRecurAMLI.c</a>	
Abstract AMLI multilevel iteration – recursive version	514
<a href="#">PreMGSolve.c</a>	
Algebraic multigrid iterations: SOLVE phase	518
<a href="#">PreSTR.c</a>	
Preconditioners for <a href="#">dSTRmat</a> matrices	521
<a href="#">SolAMG.c</a>	
AMG method as an iterative solver	527
<a href="#">SolBLC.c</a>	
Iterative solvers for <a href="#">dBLCmat</a> matrices	528
<a href="#">SolBSR.c</a>	
Iterative solvers for <a href="#">dBSRmat</a> matrices	533
<a href="#">SolCSR.c</a>	
Iterative solvers for <a href="#">dCSRmat</a> matrices	539
<a href="#">SolFAMG.c</a>	
Full AMG method as an iterative solver	546
<a href="#">SolGMGPoisson.c</a>	
GMG method as an iterative solver for Poisson Problem	548
<a href="#">SolMatFree.c</a>	
Iterative solvers using MatFree spmv operations	556
<a href="#">SolSTR.c</a>	
Iterative solvers for <a href="#">dSTRmat</a> matrices	559
<a href="#">SolWrapper.c</a>	
Wrappers for accessing functions by advanced users	563
<a href="#">XtrHBIO.c</a>	??
<a href="#">XtrMumps.c</a>	
Interface to MUMPS direct solvers	567

<a href="#">XtrPardiso.c</a>	
Interface to Intel MKL PARDISO direct solvers . . . . .	570
<a href="#">XtrSamg.c</a>	
Interface to SAMG solvers . . . . .	571
<a href="#">XtrSuperlu.c</a>	
Interface to SuperLU direct solvers . . . . .	572
<a href="#">XtrUmfpack.c</a>	
Interface to UMFPACK direct solvers . . . . .	574





## Chapter 8

# Data Structure Documentation

### 8.1 AMG\_data Struct Reference

Data for AMG solvers.

```
#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 Schwarz\\_levels](#)  
*number of levels use Schwarz smoother*
- [Schwarz\\_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 solvers.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 755 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. (BSR 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](#)

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

### 8.2.1 Detailed Description

Data for multigrid levels. (BSR format)

#### Note

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

Definition at line 151 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 solver.

```
#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 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 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 ILU<sub>t</sub>*
- [REAL ILU\\_relax](#)  
*relaxation for ILUs*
- [REAL ILU\\_permtol](#)  
*permuted if  $\text{permtol} * |a(i,j)| > |a(i,i)|$*
- [INT Schwarz\\_levels](#)  
*number of levels use Schwarz smoother*
- [INT Schwarz\\_mmsize](#)  
*maximal block size*
- [INT Schwarz\\_maxlvl](#)  
*maximal levels*
- [INT Schwarz\\_type](#)  
*type of Schwarz method*
- [INT Schwarz\\_blksolver](#)  
*type of Schwarz block solver*

### 8.3.1 Detailed Description

Parameters for AMG solver.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 616 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 115 of file fasp\_block.h.

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

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

## 8.5 block\_ivector Struct Reference

Block INT vector structure.

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

## Data Fields

- [INT brow](#)  
*row number of blocks in A, m*
- [ivector \\*\\* 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 131 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 79 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 39 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 69 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 (NNZ\*nb\*nb).

Definition at line 62 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 (or 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 (or 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 212 of file fasp.h.

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

- [fasp.h](#)

## 8.9 dCSRmat Struct Reference

Sparse matrix of REAL type in CSR 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 268 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 151 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 111 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 307 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 345 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 \\* pdir](#)
- [INT \\* edir](#)
- [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 22 of file `fasp_grid.h`.

### 8.14.2 Field Documentation

#### 8.14.2.1 `e`

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

Vertices of edges

Definition at line 25 of file `fasp_grid.h`.

#### 8.14.2.2 `edges`

```
INT edges
```

Number of edges

Definition at line 36 of file `fasp_grid.h`.

#### 8.14.2.3 `ediri`

```
INT* ediri
```

Boundary flags (0 <=> interior edge)

Definition at line 29 of file `fasp_grid.h`.

#### 8.14.2.4 efather

`INT* efather`

Father edge or triangle

Definition at line 32 of file fasp\_grid.h.

#### 8.14.2.5 p

`REAL(* p)[2]`

Coordinates of vertices

Definition at line 24 of file fasp\_grid.h.

#### 8.14.2.6 pdiri

`INT* pdiri`

Boundary flags (0 <=> interior point)

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

#### 8.14.2.7 pfather

`INT* pfather`

Father point or edge

Definition at line 31 of file fasp\_grid.h.

#### 8.14.2.8 s

`INT(* s)[3]`

Edges of triangles

Definition at line 27 of file fasp\_grid.h.



#### 8.14.2.9 t

`INT (* t) [3]`

Vertices of triangles

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

#### 8.14.2.10 tfather

`INT* tfather`

Father triangle

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

#### 8.14.2.11 triangles

`INT triangles`

Number of triangles

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

#### 8.14.2.12 vertices

`INT vertices`

Number of grid points

Definition at line 35 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 98 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 (or 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 (or IJ) format.

Coordinate Format (I,J,A)

#### Note

The starting index of A is 0.

Definition at line 242 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 181 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 130 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

- [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 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 403 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 ILU<sub>t</sub>*
- [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 377 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 \[256\]](#)
- [char workdir \[256\]](#)
- [INT problem\\_num](#)
- [SHORT solver\\_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 Schwarz\\_mmsize](#)
- [INT Schwarz\\_maxlvl](#)
- [INT Schwarz\\_type](#)
- [INT Schwarz\\_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](#)
- [INT AMG\\_coarse\\_dof](#)
- [REAL AMG\\_tol](#)
- [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\\_Schwarz\\_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](#)

### 8.21.1 Detailed Description

Input parameters.

Input parameters, reading from disk file

Definition at line 1071 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 1125 of file fasp.h.

#### 8.21.2.2 AMG\_aggressive\_level

`INT` AMG\_aggressive\_level

number of levels use aggressive coarsening

Definition at line 1130 of file fasp.h.

#### 8.21.2.3 AMG\_aggressive\_path

`INT` AMG\_aggressive\_path

number of paths used to determine strongly coupled C-set

Definition at line 1131 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 1119 of file fasp.h.

#### 8.21.2.5 AMG\_coarse\_dof

`INT` AMG\_coarse\_dof

max number of coarsest level DOF

Definition at line 1113 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 1118 of file fasp.h.

#### 8.21.2.7 AMG\_coarse\_solver

`SHORT` AMG\_coarse\_solver

coarse solver type

Definition at line 1117 of file fasp.h.

#### 8.21.2.8 AMG\_coarsening\_type

`SHORT` AMG\_coarsening\_type

coarsening type

Definition at line 1124 of file fasp.h.

#### 8.21.2.9 AMG\_cycle\_type

`SHORT AMG_cycle_type`

type of cycle

Definition at line 1106 of file fasp.h.

#### 8.21.2.10 AMG\_ILU\_levels

`SHORT AMG_ILU_levels`

how many levels use ILU smoother

Definition at line 1116 of file fasp.h.

#### 8.21.2.11 AMG\_interpolation\_type

`SHORT AMG_interpolation_type`

interpolation type

Definition at line 1126 of file fasp.h.

#### 8.21.2.12 AMG\_levels

`SHORT AMG_levels`

maximal number of levels

Definition at line 1105 of file fasp.h.

#### 8.21.2.13 AMG\_max\_aggregation

`INT AMG_max_aggregation`

max size of each aggregate

Definition at line 1137 of file fasp.h.

#### 8.21.2.14 AMG\_max\_row\_sum

`REAL` AMG\_max\_row\_sum

maximal row sum

Definition at line 1129 of file fasp.h.

#### 8.21.2.15 AMG\_maxit

`INT` AMG\_maxit

number of iterations for AMG used as preconditioner

Definition at line 1115 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 1120 of file fasp.h.

#### 8.21.2.17 AMG\_pair\_number

`INT` AMG\_pair\_number

number of pairs in matching algorithm

Definition at line 1132 of file fasp.h.

#### 8.21.2.18 AMG\_polynomial\_degree

`SHORT` AMG\_polynomial\_degree

degree of the polynomial smoother

Definition at line 1110 of file fasp.h.

**8.21.2.19 AMG\_postsmooth\_iter**

`SHORT AMG_postsmooth_iter`

number of postsmoothing

Definition at line 1112 of file fasp.h.

**8.21.2.20 AMG\_presmooth\_iter**

`SHORT AMG_presmooth_iter`

number of presmoothing

Definition at line 1111 of file fasp.h.

**8.21.2.21 AMG\_quality\_bound**

`REAL AMG_quality_bound`

threshold for pair wise aggregation

Definition at line 1133 of file fasp.h.

**8.21.2.22 AMG\_relaxation**

`REAL AMG_relaxation`

over-relaxation parameter for SOR

Definition at line 1109 of file fasp.h.

**8.21.2.23 AMG\_Schwarz\_levels**

`INT AMG_Schwarz_levels`

number of levels use Schwarz smoother

Definition at line 1121 of file fasp.h.

#### 8.21.2.24 AMG\_smooth\_filter

`SHORT` AMG\_smooth\_filter

use filter for smoothing the tentative prolongation or not

Definition at line 1139 of file fasp.h.

#### 8.21.2.25 AMG\_smooth\_order

`SHORT` AMG\_smooth\_order

order for smoothers

Definition at line 1108 of file fasp.h.

#### 8.21.2.26 AMG\_smoother

`SHORT` AMG\_smoother

type of smoother

Definition at line 1107 of file fasp.h.

#### 8.21.2.27 AMG\_strong\_coupled

`REAL` AMG\_strong\_coupled

strong coupled threshold for aggregate

Definition at line 1136 of file fasp.h.

#### 8.21.2.28 AMG\_strong\_threshold

`REAL` AMG\_strong\_threshold

strong threshold for coarsening

Definition at line 1127 of file fasp.h.

#### 8.21.2.29 AMG\_tentative\_smooth

`REAL AMG_tentative_smooth`

relaxation factor for smoothing the tentative prolongation

Definition at line 1138 of file fasp.h.

#### 8.21.2.30 AMG\_tol

`REAL AMG_tol`

tolerance for AMG if used as preconditioner

Definition at line 1114 of file fasp.h.

#### 8.21.2.31 AMG\_truncation\_threshold

`REAL AMG_truncation_threshold`

truncation factor for interpolation

Definition at line 1128 of file fasp.h.

#### 8.21.2.32 AMG\_type

`SHORT AMG_type`

Type of AMG

Definition at line 1104 of file fasp.h.

#### 8.21.2.33 ILU\_droptol

`REAL ILU_droptol`

drop tolerance

Definition at line 1093 of file fasp.h.

#### 8.21.2.34 ILU\_lfil

```
INT ILU_lfil
```

level of fill-in

Definition at line 1092 of file fasp.h.

#### 8.21.2.35 ILU\_permtol

```
REAL ILU_permtol
```

permutation tolerance

Definition at line 1095 of file fasp.h.

#### 8.21.2.36 ILU\_relax

```
REAL ILU_relax
```

scaling factor: add the sum of dropped entries to diagonal

Definition at line 1094 of file fasp.h.

#### 8.21.2.37 ILU\_type

```
SHORT ILU_type
```

ILU type for decomposition

Definition at line 1091 of file fasp.h.

#### 8.21.2.38 inifile

```
char inifile[256]
```

ini file name

Definition at line 1078 of file fasp.h.

**8.21.2.39 itsolver\_maxit**

`INT itsolver_maxit`

maximal number of iterations for iterative solvers

Definition at line 1087 of file fasp.h.

**8.21.2.40 itsolver\_tol**

`REAL itsolver_tol`

tolerance for iterative linear solver

Definition at line 1086 of file fasp.h.

**8.21.2.41 output\_type**

`SHORT output_type`

type of output stream

Definition at line 1075 of file fasp.h.

**8.21.2.42 precondition\_type**

`SHORT precondition_type`

type of preconditioner for iterative solvers

Definition at line 1084 of file fasp.h.

**8.21.2.43 print\_level**

`SHORT print_level`

print level

Definition at line 1074 of file fasp.h.



#### 8.21.2.44 problem\_num

`INT problem_num`

problem number to solve

Definition at line 1080 of file fasp.h.

#### 8.21.2.45 restart

`INT restart`

restart number used in GMRES

Definition at line 1088 of file fasp.h.

#### 8.21.2.46 Schwarz\_blksolver

`INT Schwarz_blksolver`

type of Schwarz block solver

Definition at line 1101 of file fasp.h.

#### 8.21.2.47 Schwarz\_maxlvl

`INT Schwarz_maxlvl`

maximal levels

Definition at line 1099 of file fasp.h.

#### 8.21.2.48 Schwarz\_mmsize

`INT Schwarz_mmsize`

maximal block size

Definition at line 1098 of file fasp.h.

#### 8.21.2.49 Schwarz\_type

`INT` Schwarz\_type

type of Schwarz method

Definition at line 1100 of file fasp.h.

#### 8.21.2.50 solver\_type

`SHORT` solver\_type

type of iterative solvers

Definition at line 1083 of file fasp.h.

#### 8.21.2.51 stop\_type

`SHORT` stop\_type

type of stopping criteria for iterative solvers

Definition at line 1085 of file fasp.h.

#### 8.21.2.52 workdir

`char` workdir[256]

working directory for data files

Definition at line 1079 of file fasp.h.

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

- [fasp.h](#)

## 8.22 itsolver\_param Struct Reference

Parameters passed to iterative solvers.

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

## Data Fields

- [SHORT itsolver\\_type](#)
- [SHORT precondition\\_type](#)
- [SHORT stop\\_type](#)
- [INT maxit](#)
- [REAL tol](#)
- [INT restart](#)
- [SHORT print\\_level](#)

### 8.22.1 Detailed Description

Parameters passed to iterative solvers.

Definition at line 1147 of file fasp.h.

### 8.22.2 Field Documentation

#### 8.22.2.1 itsolver\_type

[SHORT](#) itsolver\_type

solver type: see message.h

Definition at line 1149 of file fasp.h.

#### 8.22.2.2 maxit

[INT](#) maxit

max number of iterations

Definition at line 1152 of file fasp.h.

#### 8.22.2.3 precondition\_type

[SHORT](#) precondition\_type

preconditioner type: see message.h

Definition at line 1150 of file fasp.h.

#### 8.22.2.4 print\_level

`SHORT print_level`

print level: 0–10

Definition at line 1155 of file fasp.h.

#### 8.22.2.5 restart

`INT restart`

number of steps for restarting: for GMRES etc

Definition at line 1154 of file fasp.h.

#### 8.22.2.6 stop\_type

`SHORT stop_type`

stopping criteria type

Definition at line 1151 of file fasp.h.

#### 8.22.2.7 tol

`REAL tol`

convergence tolerance

Definition at line 1153 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 359 of file fasp.h.

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

- [fasp.h](#)

## 8.24 mallinfo Struct Reference

## Data Fields

- MALLINFO\_FIELD\_TYPE **arena**
- MALLINFO\_FIELD\_TYPE **ordblks**
- MALLINFO\_FIELD\_TYPE **smbblks**
- MALLINFO\_FIELD\_TYPE **hblks**
- MALLINFO\_FIELD\_TYPE **hblkhd**
- MALLINFO\_FIELD\_TYPE **usmbblks**
- MALLINFO\_FIELD\_TYPE **fsmbblks**
- MALLINFO\_FIELD\_TYPE **uordblks**
- MALLINFO\_FIELD\_TYPE **fordblks**
- MALLINFO\_FIELD\_TYPE **keepcost**

### 8.24.1 Detailed Description

Definition at line 69 of file dmalloc.h.

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

- [dmalloc.h](#)
- [malloc.c.h](#)

## 8.25 malloc\_chunk Struct Reference

### Data Fields

- `size_t` **prev\_foot**
- `size_t` **head**
- `struct malloc_chunk *` **fd**
- `struct malloc_chunk *` **bk**

### 8.25.1 Detailed Description

Definition at line 2177 of file malloc.c.h.

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

- malloc.c.h

## 8.26 malloc\_params Struct Reference

### Data Fields

- `volatile size_t` **magic**
- `size_t` **page\_size**
- `size_t` **granularity**
- `size_t` **mmap\_threshold**
- `size_t` **trim\_threshold**
- `flag_t` **default\_mflags**

### 8.26.1 Detailed Description

Definition at line 1494 of file malloc.c.h.

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

- malloc.c.h

## 8.27 malloc\_segment Struct Reference

### Data Fields

- `char *` **base**
- `size_t` **size**
- `struct malloc_segment *` **next**
- `flag_t` **sflags**

### 8.27.1 Detailed Description

Definition at line 2458 of file malloc.c.h.

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

- malloc.c.h

## 8.28 malloc\_state Struct Reference

### Data Fields

- binmap\_t **smallmap**
- binmap\_t **treemap**
- size\_t **dvsize**
- size\_t **topsize**
- char \* **least\_addr**
- [mchunkptr](#) **dv**
- [mchunkptr](#) **top**
- size\_t **trim\_check**
- size\_t **release\_checks**
- size\_t **magic**
- [mchunkptr](#) **smallbins** [(NSMALLBINS+1) \*2]
- [tbinptr](#) **treebins** [NTREEBINS]
- size\_t **footprint**
- size\_t **max\_footprint**
- flag\_t **mflags**
- [msegment](#) **seg**
- void \* **extp**
- size\_t **exts**

### 8.28.1 Detailed Description

Definition at line 2565 of file malloc.c.h.

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

- malloc.c.h

## 8.29 malloc\_tree\_chunk Struct Reference

### Data Fields

- `size_t` **prev\_foot**
- `size_t` **head**
- `struct malloc_tree_chunk *` **fd**
- `struct malloc_tree_chunk *` **bk**
- `struct malloc_tree_chunk *` **child** [2]
- `struct malloc_tree_chunk *` **parent**
- `bindex_t` **index**

### 8.29.1 Detailed Description

Definition at line 2382 of file malloc.c.h.

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

- malloc.c.h

## 8.30 Mumps\_data Struct Reference

Parameters for MUMPS interface.

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

### Data Fields

- `INT` **job**  
*work for MUMPS*

### 8.30.1 Detailed Description

Parameters for MUMPS interface.

Added on 10/10/2014

Definition at line 492 of file fasp.h.

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

- fasp.h



## 8.31 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*

#### 8.31.1 Detailed Description

Matrix-vector multiplication, replace the actual matrix.

Definition at line 1055 of file fasp.h.

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

- [fasp.h](#)

## 8.32 nedmallinfo Struct Reference

### Data Fields

- size\_t **arena**
- size\_t **ordblks**
- size\_t **smbblks**
- size\_t **hblks**
- size\_t **hblkhd**
- size\_t **usmbblks**
- size\_t **fsmbblks**
- size\_t **uordblks**
- size\_t **fordblks**
- size\_t **keepcost**

#### 8.32.1 Detailed Description

Definition at line 168 of file nedmalloc.h.

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

- [nedmalloc.h](#)

## 8.33 Pardiso\_data Struct Reference

Parameters for Intel MKL PARDISO interface.

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

### Data Fields

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

#### 8.33.1 Detailed Description

Parameters for Intel MKL PARDISO interface.

Added on 11/28/2015

Definition at line 510 of file fasp.h.

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

- [fasp.h](#)

## 8.34 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*

#### 8.34.1 Detailed Description

Preconditioner data and action.

##### Note

This is the preconditioner structure for preconditioned iterative methods.

Definition at line 1041 of file fasp.h.

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

- [fasp.h](#)

## 8.35 precondition\_block\_data Struct Reference

Data passed to the preconditioner for block preconditioning for [dBLCmat](#) format.

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

### Data Fields

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

### 8.35.1 Detailed Description

Data passed to the preconditioner for block preconditioning for [dBLCmat](#) format.

This is needed for the block preconditioner.

Definition at line 355 of file [fasp\\_block.h](#).

### 8.35.2 Field Documentation

#### 8.35.2.1 [A\\_diag](#)

[dCSRmat](#)\* [A\\_diag](#)

data for each diagonal block

Definition at line 362 of file [fasp\\_block.h](#).

#### 8.35.2.2 [Ablc](#)

[dBLCmat](#)\* [Ablc](#)

problem data, the blocks

Definition at line 360 of file [fasp\\_block.h](#).

### 8.35.2.3 amgparam

`AMG_param*` amgparam

parameters for AMG

Definition at line 374 of file fasp\_block.h.

### 8.35.2.4 LU\_diag

`void**` LU\_diag

LU decomposition for the diagonal blocks (for UMFpack)

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

### 8.35.2.5 mgl

`AMG_data**` mgl

AMG data for the diagonal blocks

Definition at line 373 of file fasp\_block.h.

### 8.35.2.6 r

`dvector` r

temp work space

Definition at line 364 of file fasp\_block.h.

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

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

## 8.36 precondition\_data Struct Reference

Data passed to the 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*

### 8.36.1 Detailed Description

Data passed to the preconditioners.

Definition at line 840 of file fasp.h.

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

- [fasp.h](#)

## 8.37 `precond_data_bsr` Struct Reference

Data passed to the preconditioners.

```
#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 aml\\_i\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [REAL \\* aml\\_i\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [REAL tentative\\_smooth](#)  
*smooth factor for smoothing the tentative prolongation*
- [SHORT nl\\_aml\\_i\\_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*

### 8.37.1 Detailed Description

Data passed to the preconditioners.

#### Note

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

Definition at line 263 of file fasp\_block.h.

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

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

## 8.38 precondition\_data\_str Struct Reference

Data passed to the preconditioner for dSTRmat matrices.

```
#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 \\* diagin](#)  
*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*

### 8.38.1 Detailed Description

Data passed to the preconditioner for [dSTRmat](#) matrices.

Definition at line 933 of file fasp.h.

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

- [fasp.h](#)

## 8.39 precondition\_diagbsr Struct Reference

Data passed to diagonal preconditioner for [dBSRmat](#) matrices.

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

### Data Fields

- [INT](#) [nb](#)  
*dimension of each sub-block*
- [dvector](#) [diag](#)  
*diagonal elements*

### 8.39.1 Detailed Description

Data passed to diagonal preconditioner for [dBSRmat](#) matrices.

#### Note

This is needed for the diagonal preconditioner.

Definition at line 246 of file `fasp_block.h`.

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

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

## 8.40 `precond_diagstr` Struct Reference

Data passed to diagonal preconditioner for [dSTRmat](#) matrices.

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

### Data Fields

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

### 8.40.1 Detailed Description

Data passed to diagonal preconditioner for [dSTRmat](#) matrices.

#### Note

This is needed for the diagonal preconditioner.

Definition at line 1025 of file `fasp.h`.

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

- [fasp.h](#)

## 8.41 precondition\_sweeping\_data Struct Reference

Data passed to the preconditioner for sweeping preconditioning.

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

### Data Fields

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

### 8.41.1 Detailed Description

Data passed to the preconditioner for sweeping preconditioning.

#### Author

Xiaozhe Hu

#### Date

05/01/2014

#### Note

This is needed for the sweeping preconditioner.

Definition at line 387 of file fasp\_block.h.

### 8.41.2 Field Documentation

#### 8.41.2.1 A

[dBLCmat \\* A](#)

problem data, the sparse matrix

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

#### 8.41.2.2 Ai

`dBLMat* Ai`

preconditioner data, the sparse matrix

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

#### 8.41.2.3 local\_A

`dCSRmat* local_A`

local stiffness matrix for each layer

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

#### 8.41.2.4 local\_index

`ivector* local_index`

local index for each layer

Definition at line 397 of file fasp\_block.h.

#### 8.41.2.5 local\_LU

`void** local_LU`

lcoal LU decomposition (for UMFpack)

Definition at line 395 of file fasp\_block.h.

#### 8.41.2.6 NumLayers

`INT NumLayers`

number of layers

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

### 8.41.2.7 `r`

`dvector r`

temporary dvector used to store and restore the residual

Definition at line 400 of file fasp\_block.h.

### 8.41.2.8 `w`

`REAL* w`

temporary work space for other usage

Definition at line 401 of file fasp\_block.h.

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

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

## 8.42 Schwarz\_data Struct Reference

Data for Schwarz methods.

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

### Data Fields

- [dCSRmat A](#)  
*pointer to the 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 Schwarz\\_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*

- [Schwarz\\_param \\* swzparam](#)

*param for Schwarz*

### 8.42.1 Detailed Description

Data for Schwarz methods.

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

Definition at line 538 of file fasp.h.

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

- [fasp.h](#)

## 8.43 Schwarz\_param Struct Reference

Parameters for Schwarz method.

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

## Data Fields

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

### 8.43.1 Detailed Description

Parameters for Schwarz method.

Added on 05/14/2012

Definition at line 467 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_array_null` (`REAL *x`)  
*Initialize an array.*
- void `fasp_array_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_array_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  $y=x$ .*
- void `fasp_array_cp_nc3` (const `REAL *x`, `REAL *y`)  
*Copy an array to the other  $y=x$ , the length is 3.*
- void `fasp_array_cp_nc5` (const `REAL *x`, `REAL *y`)  
*Copy an array to the other  $y=x$ , the length is 5.*
- void `fasp_array_cp_nc7` (const `REAL *x`, `REAL *y`)  
*Copy an array to the other  $y=x$ , the length is 7.*

### 9.1.1 Detailed Description

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

#### Note

This file contains Level-0 (Aux) functions

### 9.1.2 Function Documentation

#### 9.1.2.1 fasp\_array\_cp()

```
void fasp_array_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 173 of file AuxArray.c.

#### 9.1.2.2 fasp\_array\_cp\_nc3()

```
void fasp_array_cp_nc3 (
    const REAL * x,
    REAL * y )
```

Copy an array to the other y=x, the length is 3.

**Parameters**

<i>x</i>	Pointer to the original vector
<i>y</i>	Pointer to the destination vector

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

Special unrolled routine designed for a specific application

Definition at line 213 of file AuxArray.c.

**9.1.2.3 fasp\_array\_cp\_nc5()**

```
void fasp_array_cp_nc5 (  
    const REAL * x,  
    REAL * y )
```

Copy an array to the other y=x, the length is 5.

**Parameters**

<i>x</i>	Pointer to the original vector
<i>y</i>	Pointer to the destination vector

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

Special unrolled routine designed for a specific application

Definition at line 234 of file AuxArray.c.

#### 9.1.2.4 fasp\_array\_cp\_nc7()

```
void fasp_array_cp_nc7 (
    const REAL * x,
    REAL * y )
```

Copy an array to the other y=x, the length is 7.

##### Parameters

x	Pointer to the original vector
y	Pointer to the destination vector

##### Author

Xiaozhe Hu, Shiquan Zhang

##### Date

05/01/2010

##### Note

Special unrolled routine designed for a specific application

Definition at line 257 of file AuxArray.c.

#### 9.1.2.5 fasp\_array\_null()

```
void fasp_array_null (
    REAL * x )
```

Initialize an array.

##### Parameters

x	Pointer to the vector
---	-----------------------

##### Author

Chensong Zhang

**Date**

2010/04/03

Definition at line 31 of file AuxArray.c.

**9.1.2.6 fasp\_array\_set()**

```
void fasp_array_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 50 of file AuxArray.c.

**9.1.2.7 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 193 of file AuxArray.c.

**9.1.2.8 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 112 of file AuxArray.c.

## 9.2 AuxConvert.c File Reference

Some 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

Some utilities for encoding format conversion.

#### Note

This file contains Level-0 (Aux) functions

### 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 76 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**

x	An unsigned long integer
---	--------------------------

**Returns**

Unsigend long ineger after swapping

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 27 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**

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

**Returns**

Unsigned long integer after swapping

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 45 of file AuxConvert.c.

## 9.3 AuxGivens.c File Reference

Givens transformation.

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

**Functions**

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

### 9.3.1 Detailed Description

Givens transformation.

**Note**

This file contains Level-0 (Aux) functions

### 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 * tmp )
```

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

**Parameters**

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

**Author**

Xuehai Huang

**Date**

10/19/2008

Definition at line 30 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.*
- [INT 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.*
- [INT 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](#)

## 9.4.2 Function Documentation

### 9.4.2.1 fasp\_dbsr\_plot()

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

Write dBSR sparse matrix pattern in BMP file format.

#### Parameters

<i>A</i>	Pointer to the dBSRmat matrix
<i>filename</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 341 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 261 of file `AuxGraphics.c`.

**9.4.2.3 `fasp_dcsr_plot()`**

```
INT 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 112 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 52 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 488 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](#)

## 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 28 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 01/10/2012 Modified by Ludmil Zikatanov on 02/15/2013 Modified by Chensong Zhang on 05/10/2013: add a new input. Modified by Chensong Zhang on 03/23/2015: skip unknown keyword.

Definition at line 105 of file AuxInput.c.

## 9.6 AuxMemory.c File Reference

Memory allocation and deallocation subroutines.

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

**Functions**

- void \* [fasp\\_mem\\_calloc](#) (const [LONGLONG](#) size, const [INT](#) type)  
*1M = 1024\*1024*
- 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.*
- void [fasp\\_mem\\_usage](#) ()  
*Show total allocated memory currently.*
- [SHORT fasp\\_mem\\_check](#) (const void \*ptr, const char \*message, const [SHORT](#) ERR)  
*Check wether a point is null or not.*
- [SHORT fasp\\_mem\\_iludata\\_check](#) (const [ILU\\_data](#) \*iludata)  
*Check wether a [ILU\\_data](#) has enough work space.*

**Variables**

- unsigned [INT total\\_alloc\\_mem](#) = 0
- unsigned [INT total\\_alloc\\_count](#) = 0  
*Total allocated memory amount.*
- const [INT Million](#) = 1048576  
*Total number of allocations.*



### 9.6.1 Detailed Description

Memory allocation and deallocation subroutines.

#### Note

This file contains Level-0 (Aux) functions

### 9.6.2 Function Documentation

#### 9.6.2.1 fasp\_mem\_calloc()

```
void * fasp_mem_calloc (
    const LONGLONG size,
    const INT type )
```

1M = 1024\*1024

Allocate, initiate, and check memory

#### Parameters

<i>size</i>	Number of memory blocks
<i>type</i>	Size of memory blocks

#### Returns

Void pointer to the allocated memory

#### Author

Chensong Zhang

#### Date

2010/08/12

Modified by Chunsheng Feng on 12/20/2013 Modified by Chunsheng Feng on 07/23/2013 Modified by Chunsheng Feng on 07/30/2013 Modified by Chensong Zhang on 07/30/2013: print error if failed

Definition at line 64 of file AuxMemory.c.

### 9.6.2.2 fasp\_mem\_check()

```
SHORT fasp_mem_check (
    const void * ptr,
    const char * message,
    const SHORT ERR )
```

Check whether a point is null or not.

#### Parameters

<i>ptr</i>	Void pointer to be checked
<i>message</i>	Error message to print
<i>ERR</i>	Integer error code

#### Returns

FASP\_SUCCESS or error code

#### Author

Chensong Zhang

#### Date

11/16/2009

Definition at line 201 of file AuxMemory.c.

### 9.6.2.3 fasp\_mem\_free()

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

Free up previous allocated memory body.

#### Parameters

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

#### Returns

NULL pointer

**Author**

Chensong Zhang

**Date**

2010/12/24

Definition at line 154 of file AuxMemory.c.

**9.6.2.4 fasp\_mem\_iludata\_check()**

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

Check whether a [ILU\\_data](#) has enough work space.

**Parameters**

<i>iludata</i>	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 226 of file AuxMemory.c.

**9.6.2.5 fasp\_mem\_realloc()**

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

Reallocate, initiate, and check memory.

**Parameters**

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

**Returns**

Void pointer to the reallocated memory

**Author**

Chensong Zhang

**Date**

2010/08/12

Modified by Chunsheng Feng on 07/23/2013 Modified by Chensong Zhang on 07/30/2013: print error if failed

Definition at line 114 of file AuxMemory.c.

**9.6.2.6 fasp\_mem\_usage()**

```
void fasp_mem_usage ( )
```

Show total allocated memory currently.

**Author**

Chensong Zhang

**Date**

2010/08/12

Definition at line 179 of file AuxMemory.c.

**9.6.3 Variable Documentation**

## 9.6.3.1 total\_alloc\_count

```
unsigned INT total_alloc_count = 0
```

Total allocated memory amount.

total allocation times

Definition at line 39 of file AuxMemory.c.

## 9.6.3.2 total\_alloc\_mem

```
unsigned INT total_alloc_mem = 0
```

total allocated memory

Definition at line 38 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 `print_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.*
- void `print_amgcomplexity` (const `AMG_data` \*mgl, const `SHORT` ptrlvl)  
*Print complexities of AMG method.*
- void `print_amgcomplexity_bsr` (const `AMG_data_bsr` \*mgl, const `SHORT` ptrlvl)  
*Print complexities of AMG method for BSR matrices.*
- void `print_cputime` (const char \*message, const `REAL` cputime)  
*Print CPU walltime.*
- void `print_message` (const `INT` ptrlvl, 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

### 9.7.2 Function Documentation

#### 9.7.2.1 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 200 of file AuxMessage.c.

#### 9.7.2.2 print\_amgcomplexity()

```
void void print_amgcomplexity (
    const AMG_data * mgl,
    const SHORT prtlvl )
```

Print complexities of AMG method.

## Parameters

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

## Author

Chensong Zhang

## Date

11/16/2009

Definition at line 79 of file AuxMessage.c.

## 9.7.2.3 print\_amgcomplexity\_bsr()

```
void void print_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 123 of file AuxMessage.c.

## 9.7.2.4 print\_cputime()

```
void void print_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 166 of file AuxMessage.c.

**9.7.2.5 print\_itinfo()**

```
void print_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 36 of file AuxMessage.c.

#### 9.7.2.6 print\_message()

```
void print_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 183 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, [itsolver\\_param](#) \*itsparam, [AMG\\_param](#) \*amgparam, [ILU\\_param](#) \*iluparam, [Schwarz\\_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](#) ([itsolver\\_param](#) \*itsparam)  
*Initialize itsolver\_param.*
- void [fasp\\_param\\_ilu\\_init](#) ([ILU\\_param](#) \*iluparam)  
*Initialize ILU parameters.*
- void [fasp\\_param\\_schwarz\\_init](#) ([Schwarz\\_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\\_schwarz\\_set](#) ([Schwarz\\_param](#) \*swzparam, const [input\\_param](#) \*iniparam)  
*Set Schwarz\_param with INPUT.*
- void [fasp\\_param\\_solver\\_set](#) ([itsolver\\_param](#) \*itsparam, const [input\\_param](#) \*iniparam)  
*Set itsolver\_param with INPUT.*
- void [fasp\\_param\\_amg\\_to\\_prec](#) ([precond\\_data](#) \*pcdata, const [AMG\\_param](#) \*amgparam)  
*Set precondition\_data with AMG\_param.*
- void [fasp\\_param\\_prec\\_to\\_amg](#) ([AMG\\_param](#) \*amgparam, const [precond\\_data](#) \*pcdata)  
*Set AMG\_param with precondition\_data.*
- void [fasp\\_param\\_amg\\_to\\_prec\\_bsr](#) ([precond\\_data\\_bsr](#) \*pcdata, const [AMG\\_param](#) \*amgparam)  
*Set precondition\_data\_bsr with AMG\_param.*
- void [fasp\\_param\\_prec\\_to\\_amg\\_bsr](#) ([AMG\\_param](#) \*amgparam, const [precond\\_data\\_bsr](#) \*pcdata)  
*Set AMG\_param with precondition\_data\_bsr.*
- 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\\_schwarz\\_print](#) (const [Schwarz\\_param](#) \*param)  
*Print out Schwarz parameters.*
- void [fasp\\_param\\_solver\\_print](#) (const [itsolver\\_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](#)

## 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

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

#### Author

Chensong Zhang

#### Date

2010/04/03

Definition at line 391 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 802 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 519 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 669 of file AuxParam.c.

**9.8.2.5 fasp\_param\_amg\_to\_prec\_bsr()**

```
void fasp_param_amg_to_prec_bsr (
    precondition_data_bsr * pcddata,
    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 737 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 477 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 903 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 594 of file AuxParam.c.

## 9.8.2.9 fasp\_param\_init()

```
void fasp_param_init (
    const input_param * iniparam,
    itsolver_param * itsparam,
    AMG_param * amgparam,
    ILU_param * iluparam,
    Schwarz_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 Xiaozhe Hu (01/23/2011): initialize, then set value Modified by Chensong Zhang (09/12/2012): find a bug during debugging in VS08 Modified by Chensong Zhang (12/29/2013): rewritten

Definition at line 274 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 311 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 * pcddata )
```

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 704 of file AuxParam.c.



### 9.8.2.12 fasp\_param\_prec\_to\_amg\_bsr()

```
void fasp_param_prec_to_amg_bsr (
    AMG_param * amgparam,
    const precondition_data_bsr * pcd_data )
```

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 772 of file AuxParam.c.

### 9.8.2.13 fasp\_param\_schwarz\_init()

```
void fasp_param_schwarz_init (
    Schwarz_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 499 of file AuxParam.c.

#### 9.8.2.14 fasp\_param\_schwarz\_print()

```
void fasp_param_schwarz_print (
    const Schwarz\_param * param )
```

Print out Schwarz parameters.

##### Parameters

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

##### Author

Xiaozhe Hu

##### Date

05/22/2012

Definition at line 933 of file AuxParam.c.

#### 9.8.2.15 fasp\_param\_schwarz\_set()

```
void fasp_param_schwarz_set (
    Schwarz\_param * swzparam,
    const input\_param * iniparam )
```

Set [Schwarz\\_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 617 of file AuxParam.c.

### 9.8.2.16 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 31 of file AuxParam.c.

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

```
void fasp_param_solver_init (
    itsolver_param * itsparam )
```

Initialize [itsolver\\_param](#).

#### Parameters

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

#### Author

Chensong Zhang

#### Date

2010/03/23

Definition at line 456 of file AuxParam.c.

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

```
void fasp_param_solver_print (
    const itsolver\_param * param )
```

Print out itsolver parameters.

#### Parameters

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

#### Author

Chensong Zhang

#### Date

2011/12/20

Definition at line 962 of file AuxParam.c.

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

```
void fasp_param_solver_set (
    itsolver\_param * itsparam,
    const input\_param * iniparam )
```

Set [itsolver\\_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 639 of file AuxParam.c.

## 9.9 AuxSort.c File Reference

Subroutines for sorting, merging, removing duplicated integers.

```
#include "fasp.h"
#include "fasp_funcs.h"
```

### Functions

- [INT fasp\\_BinarySearch](#) (const [INT](#) \*list, const [INT](#) value, const [INT](#) nlist)  
*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.*
- void [fasp\\_topological\\_sorting\\_ilu](#) ([ILU\\_data](#) \*iludata)  
*Reordering vertices according to level schedule strategy.*
- void [fasp\\_multicolors\\_independent\\_set](#) ([AMG\\_data](#) \*mgl, [INT](#) gslvl)  
*Coloring vertices of adjacency graph of A.*

### 9.9.1 Detailed Description

Subroutines for sorting, merging, removing duplicated integers.

#### Note

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

### 9.9.2 Function Documentation

#### 9.9.2.1 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 247 of file AuxSort.c.

**9.9.2.2 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 328 of file AuxSort.c.

**9.9.2.3 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 209 of file AuxSort.c.

#### 9.9.2.4 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 287 of file AuxSort.c.

## 9.9.2.5 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 116 of file AuxSort.c.

**9.9.2.6 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 178 of file AuxSort.c.

### 9.9.2.7 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 83 of file AuxSort.c.

### 9.9.2.8 fasp\_BinarySearch()

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

Binary Search.

#### Parameters

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

**Returns**

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

**Author**

Chunsheng Feng

**Date**

03/01/2011

Definition at line 38 of file AuxSort.c.

**9.9.2.9 fasp\_multicolors\_independent\_set()**

```
void fasp_multicolors_independent_set (
    AMG_data * mgl,
    INT gslvl )
```

Coloring vertices of adjacency graph of A.

**Parameters**

<i>mgl</i>	Pointer to input matrix
<i>gslvl</i>	Used to specify levels of AMG using multicolor smoothing

**Author**

Zheng Li, Chunsheng Feng

**Date**

12/04/2016

Definition at line 444 of file AuxSort.c.

**9.9.2.10 fasp\_topological\_sorting\_ilu()**

```
void fasp_topological_sorting_ilu (
    ILU_data * iludata )
```

Reordering vertices according to level schedule strategy.

## Parameters

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

## Author

Zheng Li, Chensong Zhang

## Date

12/04/2016

Definition at line 362 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

## 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

<i>procid</i>	Index of thread
<i>nprocs</i>	Number of threads
<i>n</i>	Total workload
<i>start</i>	Pointer to the begin of each thread in total workload
<i>end</i>	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 86 of file AuxThreads.c.

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

```
void fasp_set_GS_threads (
    const INT threads,
    const INT its )
```

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

#### Parameters

<i>threads</i>	Total threads of solver
<i>its</i>	Current its of the Krylov methods

**Author**

Feng Chunsheng, Yue Xiaoqiang

**Date**

03/20/2011

Definition at line 126 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 110 of file AuxThreads.c.

#### 9.10.3.2 THDs\_CPR\_gGS

```
INT THDs_CPR_gGS =0
```

global matrix GS smoothing threads

Definition at line 112 of file AuxThreads.c.

#### 9.10.3.3 THDs\_CPR\_IGS

```
INT THDs_CPR_IGS =0
```

reservoir GS smoothing threads

Definition at line 111 of file AuxThreads.c.

## 9.11 AuxTiming.c File Reference

Timing subroutines.

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

#### 9.11.2 Function Documentation

##### 9.11.2.1 fasp\_gettime()

```
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 30 of file AuxTiming.c.



## 9.12 AuxVector.c File Reference

Simple operations for vectors.

```
#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\\_null](#) ([dvector](#) \*x)  
*Initialize dvector.*
- void [fasp\\_dvec\\_rand](#) (const [INT](#) n, [dvector](#) \*x)  
*Generate random REAL vector in the range from 0 to 1.*
- void [fasp\\_dvec\\_set](#) ([INT](#) n, [dvector](#) \*x, [REAL](#) val)  
*Initialize dvector x[i]=val for i=0:n-1.*
- void [fasp\\_ivec\\_set](#) (const [INT](#) m, [ivector](#) \*u)  
*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 operations for vectors.

#### Note

This file contains Level-0 (Aux) functions

## 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

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

#### Author

Chensong Zhang

#### Date

2010/04/06

Definition at line 99 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 344 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 56 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 139 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 33 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 367 of file AuxVector.c.

**9.12.2.7 fasp\_dvec\_null()**

```
void fasp_dvec_null (  
    dvector * x )
```

Initialize dvector.

**Parameters**

<i>x</i>	Pointer to dvector which needs to be initialized
----------	--

**Author**

Chensong Zhang

**Date**

2010/04/03

Definition at line 177 of file AuxVector.c.

**9.12.2.8 fasp\_dvec\_rand()**

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

Generate 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 203 of file AuxVector.c.

## 9.12.2.9 fasp\_dvec\_set()

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

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

## Parameters

<i>n</i>	Number of variables
<i>x</i>	Pointer to dvector
<i>val</i>	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 233 of file AuxVector.c.

## 9.12.2.10 fasp\_dvec\_symdiagscale()

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

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

## Parameters

<i>b</i>	Pointer to dvector
<i>diag</i>	Pointer to dvector: the diagonal entries

## Author

Xiaozhe Hu

**Date**

01/31/2011

Definition at line 420 of file AuxVector.c.

**9.12.2.11 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 119 of file AuxVector.c.

**9.12.2.12 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 78 of file AuxVector.c.

**9.12.2.13 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 159 of file AuxVector.c.

**9.12.2.14 fasp\_ivec\_set()**

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

Set ivector value to be m.

## Parameters

<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

## Date

05/23/2012

Definition at line 302 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_array_ax` (const `INT` n, const `REAL` a, `REAL` \*x)  
 $x = a * x$
- void `fasp_blas_array_axpy` (const `INT` n, const `REAL` a, const `REAL` \*x, `REAL` \*y)  
 $y = a * x + y$
- void `fasp_blas_array_axpyz` (const `INT` n, const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)  
 $z = a * x + y$
- void `fasp_blas_array_axpby` (const `INT` n, const `REAL` a, const `REAL` \*x, const `REAL` b, `REAL` \*y)  
 $y = a * x + b * y$
- `REAL fasp_blas_array_dotprod` (const `INT` n, const `REAL` \*x, const `REAL` \*y)  
*Inner product of two arrays (x,y)*
- `REAL fasp_blas_array_norm1` (const `INT` n, const `REAL` \*x)  
*L1 norm of array x.*
- `REAL fasp_blas_array_norm2` (const `INT` n, const `REAL` \*x)  
*L2 norm of array x.*
- `REAL fasp_blas_array_norminf` (const `INT` n, const `REAL` \*x)  
*Linf norm of array x.*

### 9.13.1 Detailed Description

BLAS1 operations for arrays.

#### Note

This file contains Level-1 (Bla) functions.

### 9.13.2 Function Documentation

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

```
void fasp_blas_array_ax (  
    const INT n,  
    const REAL a,  
    REAL * x ) [inline]
```

$x = a * x$

#### Parameters

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

#### Author

Chensong Zhang

#### Date

07/01/209

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

#### Note

$x$  is reused to store the resulting array.

Definition at line 37 of file BlaArray.c.

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

```
void fasp_blas_array_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

#### Author

Chensong Zhang

#### Date

07/01/2009

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

#### Note

$y$  is reused to store the resulting array.

Definition at line 223 of file BlaArray.c.

### 9.13.2.3 fasp\_blas\_array\_axpy()

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

$y = a*x + y$

## Parameters

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

## Author

Chensong Zhang

## Date

07/01/2009

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

## Note

$y$  is reused to store the resulting array.

Definition at line 90 of file BlaArray.c.

## 9.13.2.4 fasp\_blas\_array\_axpyz()

```
void fasp_blas_array_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 172 of file BlaArray.c.

**9.13.2.5 fasp\_blas\_array\_dotprod()**

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

Inner product of two arrays (x,y)

**Parameters**

$n$	Number of variables
$x$	Pointer to x
$y$	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 272 of file BlaArray.c.

#### 9.13.2.6 fasp\_blas\_array\_norm1()

```
REAL fasp_blas_array_norm1 (  
    const INT n,  
    const REAL * x )
```

L1 norm of array x.

##### Parameters

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

##### Returns

L1 norm of x

##### Author

Chensong Zhang

##### Date

07/01/209

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

Definition at line 313 of file BlaArray.c.

#### 9.13.2.7 fasp\_blas\_array\_norm2()

```
REAL fasp_blas_array_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**

07/01/209

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

Definition at line 343 of file BlaArray.c.

**9.13.2.8 fasp\_blas\_array\_norminf()**

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

Linf norm of array x.

**Parameters**

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

**Returns**

L\_inf norm of x

**Author**

Chensong Zhang

**Date**

07/01/209

Modified by Chunsheng Feng, Zheng Li on 06/28/2012

Definition at line 372 of file BlaArray.c.



## 9.14 BlaEigen.c File Reference

Subroutines for computing the extreme eigenvalues.

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

### Functions

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

### 9.14.1 Detailed Description

Subroutines for computing the extreme eigenvalues.

#### Note

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

### 9.14.2 Function Documentation

#### 9.14.2.1 fasp\_dcsr\_eig()

```
REAL fasp_dcsr_eig (
    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 32 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\\_dcsr\\_dcsr](#) (const [dCSRmat](#) \*A)  
*Convert a dCSRmat into a dCSRmat.*
- [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](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), and [BlaSparseCSRL.c](#).

### 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 <a href="#">dBLCmat</a> matrix
-----------	---

#### Returns

[dCSRmat](#) matrix if succeed, NULL if fail

#### Author

Shiquan Zhang

#### Date

08/10/2010

Definition at line 290 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**

$B$	Pointer to <a href="#">dBSRmat</a> matrix
-----	---

**Returns**

Pointer to [dCOOmat](#) matrix

**Author**

Zhiyang Zhou

**Date**

2010/10/26

Definition at line 941 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**

$B$	Pointer to <a href="#">dBSRmat</a> 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 493 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 31 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 719 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

## Date

10/12/2012

Definition at line 79 of file BlaFormat.c.

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

```
dCSRLmat * fasp_format_dcsrl_dcsr (  
    const dCSRmat * A )
```

Convert a [dCSRmat](#) into a [dCSRLmat](#).

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRLmat</a> matrix
----------	--

## Returns

Pointer to [dCSRLmat](#) matrix

## Author

Zhiyang Zhou

## Date

2011/01/07

Definition at line 359 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 <a href="#">dSTRmat</a> matrix
----------	---

**Returns**

[dBSRmat](#) matrix

**Author**

Zhiyang Zhou

**Date**

2010/10/26

Definition at line 837 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 115 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 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 \*rowptr, 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 version by Chunsheng Feng, 09/03/2016

### 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 <i>lfil</i> elements (excluding the diagonal element). <i>lfil</i> must be .ge. 0.
<i>alu,jlu</i>	matrix stored in Modified Sparse Row (MSR) format containing the L and U factors together. The diagonal (stored in <i>alu</i> (1:n) ) is inverted. Each i-th row of the <i>alu,jlu</i> 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 <i>alu,jlu</i> .
<i>iwk</i>	integer. The minimum length of arrays <i>alu</i> , <i>jlu</i> , and <i>levs</i> .
<i>ierr</i>	integer pointer. Return error message with the following meaning. 0 -> successful return. >0 -> zero pivot encountered at step number <i>ierr</i> . -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 <i>al</i> . -3 -> The matrix U overflows the array <i>alu</i> . -4 -> Illegal value for <i>lfil</i> . -5 -> zero row encountered.
<i>nzlu</i>	integer pointer. Return number of nonzero entries for <i>alu</i> and <i>jlu</i>

## Note

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

## Author

Chunsheng Feng

## Date

09/06/2016

Definition at line 66 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 <i>lfil</i> elements (excluding the diagonal element). <i>lfil</i> 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,jlu</i>	matrix stored in Modified Sparse Row (MSR) format containing the L and U factors together. The diagonal (stored in <i>alu</i> (1:n) ) is inverted. Each i-th row of the <i>alu,jlu</i> matrix contains the i-th row of L (excluding the diagonal entry=1) followed by the i-th row of U.
<i>iwk</i>	integer. The lengths of arrays <i>alu</i> and <i>jlu</i> . If the arrays are not big enough to store the ILU factorizations, <i>ilut</i> 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 <i>ierr</i> . -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 <i>al</i> . -3 -> The matrix U overflows the array <i>alu</i> . -4 -> Illegal value for <i>lfil</i> . -5 -> zero row encountered.
<i>nz</i>	integer pointer. Return number of nonzero entries for <i>alu</i> and <i>jlu</i>

## Note

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

## Author

Chunsheng Feng

## Date

09/06/2016

Definition at line 458 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 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,jlu</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>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 893 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



**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.  $ijlu$  = 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:  $ierr = -i \rightarrow$  near zero pivot in step  $i$   $ierr = 0 \rightarrow$  all's OK  $ierr = 1 \rightarrow$  not enough storage; check  $mneed$ .  $ierr = 2 \rightarrow$  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.  $rowll$  = 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 1359 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\\_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\\_omp](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Multi-threads parallel ILU decoposition of a BSR matrix A based on graph coloring.*
- [SHORT fasp\\_ilu\\_dbsr\\_setup\\_mc\\_omp](#) ([dBSRmat](#) \*A, [dCSRmat](#) \*Ap, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Multi-threads parallel 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](#), [AuxSort.c](#), [AuxTiming.c](#), [BlaSmallMatInv.c](#), [BlaiLU.c](#), [BlaSmallMat.c](#), [BlaSparseBSR.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), and [PreDataInit.c](#)

### 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.

Definition at line 44 of file BlalLUSetupBSR.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 1, 2, 3 nb (Zheng)

Definition at line 845 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-threads parallel 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 and provide 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)

Definition at line 1081 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-threads parallel 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)

Definition at line 969 of file [BlaILUSetupBSR.c](#).

## 9.18 BlaILUSetupCSR.c File Reference

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

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.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](#), [BlaILU.c](#), [BlaSparseCSR.c](#), and [PreDataInit.c](#)

## 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

Definition at line 33 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](#), [BlaSmallMatInv.c](#), [BlaSmallMat.c](#), [BlaSparseSTR.c](#), and [BlaArray.c](#)

### 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 33 of file [BlaILUSetupSTR.c](#).

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

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

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

## Parameters

<i>A</i>	Pointer to oringinal 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 328 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"
```

## Functions

- void `fasp_dcsrvec1_read` (const char \*filename, `dCSRmat` \*A, `dvector` \*b)  
*Read A and b from a SINGLE disk file.*
- void `fasp_dcsrvec2_read` (const char \*filemat, const char \*filerhs, `dCSRmat` \*A, `dvector` \*b)  
*Read A and b from two 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_dcoo1_read` (const char \*filename, `dCOOmat` \*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)  
*Read b from matrix disk file.*
- 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_dcsrvec1_write` (const char \*filename, `dCSRmat` \*A, `dvector` \*b)  
*Write A and b to a SINGLE disk file.*
- void `fasp_dcsrvec2_write` (const char \*filemat, const char \*filerhs, `dCSRmat` \*A, `dvector` \*b)  
*Write A and b to two 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_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\\_print](#) (const [dBSRmat](#) \*A)
- Print out a dBSRmat 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 from both ASCII and 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](#), and [BlaFormat.c](#)

## 9.20.2 Function Documentation

### 9.20.2.1 fasp\_dbsr\_print()

```
void fasp_dbsr_print (
    const dBSRmat * A )
```

Print out a [dBSRmat](#) matrix in coordinate format.

#### Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix A
----------	---

#### Author

Ziteng Wang

#### Date

12/24/2012

Modified by Chunsheng Feng on 11/16/2013

Definition at line 1454 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 700 of file BlalO.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 \ref fasp\_dbsr\_read.

**Author**

Shiquan Zhang

**Date**

10/29/2010

Definition at line 1207 of file BlalO.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 1490 of file BlalO.c.

#### 9.20.2.5 fasp\_dcoo1\_read()

```
void fasp_dcoo1_read (
    const char * filename,
    dCOOmat * 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 COO 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

difference between fasp\_dcoo\_read and this function is this function do not change to CSR format

**Author**

Xiaozhe Hu

**Date**

03/24/2013

Definition at line 379 of file BlalO.c.

**9.20.2.6 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 dCOOmat matrix A
----------	---------------------------------

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1432 of file BlalO.c.

**9.20.2.7 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 328 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 fashin and converts the matrix to [dCSRmat](#) format.

**Author**

Xiaozhe Hu

## Date

04/01/2014

Definition at line 429 of file BlalO.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 [\ref 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 1108 of file BlalO.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 <a href="#">dCSRmat</a> matrix A
----------	---

## Author

Xuehai Huang

## Date

03/29/2009

Definition at line 1410 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 267 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 <a href="#">dCSRmat</a> matrix <i>A</i>

## Author

Chunsheng Feng

## Date

11/14/2013

Definition at line 1540 of file BlaiO.c.

## 9.20.2.13 fasp\_dcsrvec1\_read()

```
void fasp_dcsrvec1_read (
    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 96 of file BlalO.c.

**9.20.2.14 fasp\_dcsrvec1\_write()**

```
void fasp_dcsrvec1_write (
    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 959 of file BlalO.c.

**9.20.2.15 fasp\_dcsrvec2\_read()**

```
void fasp_dcsrvec2_read (
    const char * filemat,
    const char * filerhs,
    dCSRmat * A,
    dvector * b )
```

Read A and b from two 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 2011/03/01 Modified by Chensong Zhang on 2012/01/05

Definition at line 188 of file BlalO.c.

## 9.20.2.16 fasp\_dcsrvec2\_write()

```
void fasp_dcsrvec2_write (
    const char * filemat,
    const char * filerhs,
    dCSRmat * A,
    dvector * b )
```

Write A and b to two 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 1037 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 dCSR↵  
Rmat 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 481 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 543 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**

<b>A</b>	Pointer to the <b>dSTRmat</b> matrix A
----------	--

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1579 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 620 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 \ref fasp\_dstr\_read.

**Author**

Shiquan Zhang

**Date**

03/29/2010

Definition at line 1148 of file BlalO.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 1367 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<sub>j</sub>, j=0:nrow-1

**Author**

Chensong Zhang

**Date**

03/29/2009

Definition at line 819 of file BlaiO.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 1262 of file BlaiO.c.

#### 9.20.2.25 fasp\_dvecind\_read()

```
void fasp_dvecind_read (
    const char * filename,
    dvector * 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

Because the index is given, order is not important!

##### Author

Chensong Zhang

##### Date

03/29/2009

Definition at line 769 of file BlaiO.c.

#### 9.20.2.26 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 1298 of file BlaiO.c.

**9.20.2.27 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

**Author**

Xiaoehe Hu

**Date**

05/30/2014

Definition at line 2069 of file BlaiO.c.

#### 9.20.2.28 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 1389 of file BlalO.c.

#### 9.20.2.29 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\_j, j=0:nrow-1

**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 909 of file BlaiO.c.

**9.20.2.30 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 1333 of file BlaiO.c.

**9.20.2.31 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 869 of file BlaiO.c.

**9.20.2.32 fasp\_matrix\_read()**

```
fasp_matrix_read (
    const char * filemat,
    void * A )
```

Read matrix from different kinds of formats from both ASCII and binary files.

**Parameters**

<i>filemat</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
- dlength % 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 1613 of file BlaiO.c.

**9.20.2.33 fasp\_matrix\_read\_bin()**

```
void fasp_matrix_read_bin (
    const char * filemat,
    void * A )
```

Read matrix in binary format.

**Parameters**

<i>filemat</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 1714 of file BlaiO.c.

#### 9.20.2.34 fasp\_matrix\_write()

```
fasp_matrix_write (
    const char * filemat,
    void * A,
    const INT flag )
```

write matrix from different kinds of formats from both ASCII and binary files

##### Parameters

<i>filemat</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 1788 of file BlalO.c.

#### 9.20.2.35 fasp\_vector\_read()

```
fasp_vector_read (
    const char * filerhs,
    void * b )
```

Read RHS vector from different kinds of formats from both ASCII and 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 1882 of file BlaiO.c.

## 9.20.2.36 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 1980 of file BlaiO.c.

### 9.20.3 Variable Documentation

#### 9.20.3.1 dlength

`INT dlength`

Length of REAL in byte

Definition at line 18 of file BlaiO.c.

#### 9.20.3.2 ilength

`INT ilength`

Length of INT in byte

Definition at line 17 of file BlaiO.c.

## 9.21 BlaOrderingCSR.c File Reference

Subroutines for 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

Subroutines for generating ordering using algebraic information.

#### Note

This file contains Level-1 (Bla) functions.

### 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 32 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 82 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_schwarz_setup` (`Schwarz_data` \*Schwarz, `Schwarz_param` \*param)  
*Setup phase for the Schwarz methods.*
- `void fasp_dcsr_schwarz_forward_smoother` (`Schwarz_data` \*Schwarz, `Schwarz_param` \*param, `dvector` \*x, `dvector` \*b)  
*Schwarz smoother: forward sweep.*
- `void fasp_dcsr_schwarz_backward_smoother` (`Schwarz_data` \*Schwarz, `Schwarz_param` \*param, `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](#)

### 9.22.2 Function Documentation

#### 9.22.2.1 fasp\_dcsr\_schwarz\_backward\_smoother()

```
void fasp_dcsr_schwarz_backward_smoother (
    Schwarz_data * Schwarz,
    Schwarz_param * param,
    dvector * x,
    dvector * b )
```

Schwarz smoother: backward sweep.

#### Parameters

<i>Schwarz</i>	Pointer to the Schwarz data
<i>param</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 321 of file BlaSchwarzSetup.c.

**9.22.2.2 fasp\_dcsr\_schwarz\_forward\_smoother()**

```
void fasp_dcsr_schwarz_forward_smoother (
    Schwarz_data * Schwarz,
    Schwarz_param * param,
    dvector * x,
    dvector * b )
```

Schwarz smoother: forward sweep.

**Parameters**

<i>Schwarz</i>	Pointer to the Schwarz data
<i>param</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 211 of file BlaSchwarzSetup.c.

**9.22.2.3 fasp\_schwarz\_setup()**

```
INT fasp_schwarz_setup (
    Schwarz_data * Schwarz,
    Schwarz_param * param )
```

Setup phase for the Schwarz methods.

## Parameters

<i>Schwarz</i>	Pointer to the Schwarz data
<i>param</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 42 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_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_mul_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_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.*

  - 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.*

  - 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.*

  - 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.*

  - void `fasp_blas_array_axpyz_nc2` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)

$z = a*x + y$
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpyz_nc3` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)

$z = a*x + y$
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpyz_nc5` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)

$z = a*x + y$
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpyz_nc7` (const `REAL` a, const `REAL` \*x, const `REAL` \*y, `REAL` \*z)

$z = a*x + y$
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpy_nc2` (const `REAL` a, const `REAL` \*x, `REAL` \*y)

$y = a*x + y$ , the length of x and y is 2
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpy_nc3` (const `REAL` a, const `REAL` \*x, `REAL` \*y)

$y = a*x + y$ , the length of x and y is 3
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpy_nc5` (const `REAL` a, const `REAL` \*x, `REAL` \*y)

$y = a*x + y$ , the length of x and y is 5
- Compute the matrix product of two small full matrices a and b, stored in c.*

  - void `fasp_blas_array_axpy_nc7` (const `REAL` a, const `REAL` \*x, `REAL` \*y)

$y = a*x + y$ , the length of x and y is 7
- Compute  $y := y + Ax$ , where 'A' is a 2\*2 dense matrix.*

  - void `fasp_blas_smat_yAx_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_yAx_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_yAx_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_yAx` (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_ymAx_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_ymAx_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_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)

$y := \alpha * A * x + \beta * y$ .



### 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](#)

#### 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\_array\_axpy\_nc2()

```
void fasp_blas_array_axpy_nc2 (  
    const REAL a,  
    const REAL * x,  
    REAL * y )
```

$y = a*x + y$ , the length of  $x$  and  $y$  is 2

#### Parameters

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

#### Author

Xiaozhe Hu

#### Date

18/11/2011

Definition at line 694 of file BlaSmallMat.c.

### 9.23.2.2 fasp\_blas\_array\_axpy\_nc3()

```
void fasp_blas_array_axpy_nc3 (
    const REAL a,
    const REAL * x,
    REAL * y )
```

$y = a*x + y$ , the length of  $x$  and  $y$  is 3

#### 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 717 of file BlaSmallMat.c.

### 9.23.2.3 fasp\_blas\_array\_axpy\_nc5()

```
void fasp_blas_array_axpy_nc5 (
    const REAL a,
    const REAL * x,
    REAL * y )
```

$y = a*x + y$ , the length of  $x$  and  $y$  is 5

#### 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 746 of file BlasSmallMat.c.

**9.23.2.4 fasp\_blas\_array\_axpy\_nc7()**

```
void fasp_blas_array_axpy_nc7 (
    const REAL a,
    const REAL * x,
    REAL * y )
```

$y = a*x + y$ , the length of  $x$  and  $y$  is 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 793 of file BlasSmallMat.c.

**9.23.2.5 fasp\_blas\_array\_axpyz\_nc2()**

```
void fasp_blas_array_axpyz_nc2 (
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

$z = a*x + y$

**Parameters**

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array 1
<i>y</i>	Pointer to the original array 2
<i>z</i>	Pointer to the destination array

**Author**

Xiaozhe Hu

**Date**

18/11/2011

**Note**

*z* is the third array and the length of *x*, *y* and *z* is 2

Definition at line 506 of file BlaSmallMat.c.

**9.23.2.6 fasp\_blas\_array\_axpyz\_nc3()**

```
void fasp_blas_array_axpyz_nc3 (
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

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

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array 1
<i>y</i>	Pointer to the original array 2
<i>z</i>	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

z is the third array and the length of x, y and z is 3

Definition at line 534 of file BlasSmallMat.c.

**9.23.2.7 fasp\_blas\_array\_axpyz\_nc5()**

```
void fasp_blas_array_axpyz_nc5 (  
    const REAL a,  
    const REAL * x,  
    const REAL * y,  
    REAL * z )
```

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

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array 1
<i>y</i>	Pointer to the original array 2
<i>z</i>	Pointer to the destination array

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

z is the third array and the length of x, y and z is 5

Definition at line 568 of file BlasSmallMat.c.

### 9.23.2.8 fasp\_blas\_array\_axpyz\_nc7()

```
void fasp_blas_array_axpyz_nc7 (
    const REAL a,
    const REAL * x,
    const REAL * y,
    REAL * z )
```

$z = a*x + y$

#### Parameters

<i>a</i>	REAL factor a
<i>x</i>	Pointer to the original array 1
<i>y</i>	Pointer to the original array 2
<i>z</i>	Pointer to the destination array

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

05/01/2010

#### Note

*z* is the third array and the length of *x*, *y* and *z* is 7

Definition at line 620 of file BlaSmallMat.c.

### 9.23.2.9 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 1289 of file BlasSmallMat.c.

## 9.23.2.10 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*n matrix
<i>b</i>	Pointer to the REAL array which stands a n*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*n matrix

## Author

Xiaozhe Hu, Chensong Zhang

## Date

05/26/2014

Definition at line 60 of file BlaSmallMat.c.

## 9.23.2.11 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 $n*n$ matrix
<i>n</i>	Dimension of the matrix
<i>alpha</i>	Scalar

## Author

Xiaozhe Hu, Chensong Zhang

## Date

05/26/2014

Definition at line 32 of file BlaSmallMat.c.

## 9.23.2.12 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

Definition at line 453 of file BlaSmallMat.c.

**9.23.2.13 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 238 of file BlaSmallMat.c.

#### 9.23.2.14 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 267 of file BlaSmallMat.c.

#### 9.23.2.15 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 304 of file BlasSmallMat.c.

**9.23.2.16 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 363 of file BlasSmallMat.c.

**9.23.2.17 fasp\_blas\_smat\_mxv()**

```
void fasp_blas_smat_mxv (
    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

Definition at line 188 of file BlaSmallMat.c.

**9.23.2.18 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 88 of file BlaSmallMat.c.

## 9.23.2.19 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 110 of file BlasSmallMat.c.

## 9.23.2.20 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 133 of file BlaSmallMat.c.

**9.23.2.21 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 159 of file BlaSmallMat.c.

**9.23.2.22 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\*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 1187 of file BlasSmallMat.c.

## 9.23.2.23 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 1072 of file BlaSmallMat.c.

**9.23.2.24 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 1098 of file BlaSmallMat.c.

**9.23.2.25 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 1125 of file BlasSmallMat.c.

**9.23.2.26 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 1154 of file BlaSmallMat.c.

**9.23.2.27 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 972 of file BlaSmallMat.c.

**9.23.2.28 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*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 866 of file BlasSmallMat.c.

## 9.23.2.29 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 890 of file BlasSmallMat.c.

## 9.23.2.30 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

A	Pointer to the 5*5 dense matrix
x	Pointer to the REAL array with length 5
y	Pointer to the REAL array with length 5

## Author

Zhiyang Zhou, Xiaozhe Hu, Chensong Zhang

## Date

2010/10/25

Definition at line 914 of file BlaSmallMat.c.

## 9.23.2.31 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

A	Pointer to the 7*7 dense matrix
x	Pointer to the REAL array with length 7
y	Pointer to the REAL array with length 7

## Author

Zhiyang Zhou, Xiaozhe Hu, Chensong Zhang

Date

2010/10/25

Definition at line 940 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.*
- void [fasp\\_smat\\_invp\\_nc](#) (REAL \*a, const INT n)  
*Compute the inverse of a matrix using Gauss Elimination with Pivoting.*
- INT [fasp\\_smat\\_inv](#) (REAL \*a, const INT n)  
*Compute the inverse matrix of a small full matrix a.*
- REAL [fasp\\_smat\\_Linfinity](#) (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](#)

### 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 12 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\*n full matrix to be a identity.

#### Parameters

<i>a</i>	Pointer to the REAL vector which stands for a n*n full matrix
<i>n</i>	Size of full matrix
<i>n2</i>	Length of the REAL vector which stores the n*n full matrix

**Author**

Xiaozhe Hu

**Date**

2010/12/25

Definition at line 713 of file BlaSmallMatInv.c.

**9.24.3.2 fasp\_smat\_identity\_nc2()**

```
void fasp_smat_identity_nc2 (  
    REAL * a )
```

Set a 2\*2 full matrix to be a identity.

**Parameters**

<i>a</i>	Pointer to the REAL vector which stands for a 2*2 full matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

2011/11/18

Definition at line 633 of file BlaSmallMatInv.c.

**9.24.3.3 fasp\_smat\_identity\_nc3()**

```
void fasp_smat_identity_nc3 (  
    REAL * a )
```

Set a 3\*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 650 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 667 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 688 of file BlaSmallMatInv.c.

**9.24.3.6 fasp\_smat\_inv()**

```
INT 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 559 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 $n \times n$ matrix
<i>n</i>	Dimension of the matrix

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

Definition at line 410 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  $2 \times 2$  full matrix A (in place)

**Parameters**

<i>a</i>	Pointer to the REAL array which stands a $2 \times 2$ matrix
----------	--

**Author**

Xiaozhe Hu

**Date**

18/11/2011

Definition at line 28 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  $3 \times 3$  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 64 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 119 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 178 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 394 of file BlaSmallMatInv.c.

**9.24.3.13 fasp\_smat\_invp\_nc()**

```
void 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 Recipies in C"!

Definition at line 477 of file BlaSmallMatInv.c.

## 9.24.3.14 fasp\_smat\_Linfinity()

```
REAL fasp_smat_Linfinity (  
    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 600 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.

### 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 47 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 119 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.*
- [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.*

### 9.26.1 Detailed Description

Sparse matrix block operations.

**Note**

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



## 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

<i>A</i>	Pointer to the <a href="#">dBLCmat</a> matrix
----------	---

#### Author

Xiaozhe Hu

#### Date

04/18/2014

Definition at line 33 of file BlaSparseBLC.c.

### 9.26.2.2 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
<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 69 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\\_null](#) ([dBSRmat](#) \*A)  
*Initialize sparse matrix on structured grid.*
- 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\\_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\\_getdiagin](#) (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.

## 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](#), [BlaSmallMatInv.c](#), and [BlaSmallMat.c](#)

## 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 <code>dBSRmat</code> matrix

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 90 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 dBSRmat matrix
<i>B</i>	Pointer to the dBSRmat matrix

**Author**

Xiaozhe Hu

**Date**

08/07/2011

Definition at line 184 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 39 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

Definition at line 501 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 665 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 767 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 1125 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 dBSRmat 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 1454 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 <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

Zheng Li, Xiaozhe Hu

## Date

06/17/2014

Definition at line 1683 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 295 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**

<b>A</b>	Pointer to the <a href="#">dBSRmat</a> matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 136 of file BlaSparseBSR.c.

## 9.27.2.12 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 1416 of file BlaSparseBSR.c.

## 9.27.2.13 fasp\_dbsr\_getdiaginv()

```
dvector fasp_dbsr_getdiaginv (
    const dBSRmat * A )
```

Get  $D^{-1}$  of matrix A.

## Parameters

<i>A</i>	Pointer to the dBSRmat matrix
----------	-------------------------------

**Author**

Xiaozhe Hu

**Date**

02/19/2013

**Note**

Works for general nb (Xiaozhe)

Definition at line 395 of file BlaSparseBSR.c.

**9.27.2.14 fasp\_dbsr\_null()**

```
void fasp_dbsr_null (
    dBSRmat * A )
```

Initialize sparse matrix on structured grid.

**Parameters**

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
----------	---

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 161 of file BlaSparseBSR.c.

**9.27.2.15 fasp\_dbsr\_perm()**

```
dBSRmat fasp_dbsr_perm (
    const dBSRmat * A,
    const 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 the given ordering

## Returns

The new ordered [dCSRmat](#) 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 1884 of file BlaSparseBSR.c.

## 9.27.2.16 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 211 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 whether 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 diagonal 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.*

### 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](#)

### 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

<i>A</i>	Pointer to the matrix in <a href="#">dCSRmat</a> format
----------	---

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 278 of file BlaSparseCheck.c.

## 9.28.2.2 fasp\_check\_diagdom()

```
INT fasp_check_diagdom (
    const dCSRmat * A )
```

Check whether a matrix is diagonal 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 diagonal dominant

## Note

The routine checks whether the sparse matrix is diagonal dominant on every row. It will print out the percentage of the rows which are diagonal dominant and which are not; the routine will return the number of the rows which are diagonal dominant.

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 111 of file BlaSparseCheck.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 30 of file BlaSparseCheck.c.

### 9.28.2.4 fasp\_check\_diagzero()

```
SHORT fasp_check_diagzero (
    const dCSRmat * A )
```

Check whether 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 67 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

<a href="#">A</a>	Pointer to the matrix in <a href="#">iCSRmat</a> format
-------------------	---

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 315 of file BlaSparseCheck.c.

## 9.28.2.6 fasp\_check\_symm()

```
INT fasp_check_symm (
    const dCSRmat * A )
```

Check symmetry of a sparse matrix of CSR format.

## Parameters

<a href="#">A</a>	Pointer to the <a href="#">dCSRmat</a> 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 156 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](#)

## 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 dCSRmat matrix

#### Author

Xiaozhe Hu

#### Date

03/25/2013

Definition at line 65 of file BlasSparseCOO.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 37 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**

<code>A</code>	Pointer to the <code>dCOOmat</code> matrix
----------------	--

**Author**

Chensong Zhang

**Date**

2010/04/03

Definition at line 97 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 119 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.*
- void [fasp\\_dcsr\\_null](#) ([dCSRmat](#) \*A)  
*Initialize CSR sparse matrix.*
- void [fasp\\_icsr\\_null](#) ([iCSRmat](#) \*A)  
*Initialize CSR sparse matrix.*
- [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)  
*Find transpose of dCSRmat matrix A.*
- 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  $abs(a_{ij}) \leq dtol$ .*
- **SHORT fasp\_dcsr\_compress\_inplace** (dCSRmat \*A, const REAL dtol)  
*Compress a CSR matrix A IN PLACE by dropping small entries  $abs(a_{ij}) \leq dtol$ .*
- 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 x 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](#), [BlaSparseCSR.c](#), [BlaSparseUtil.c](#), and [BlaArray.c](#)

### 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
<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 129 of file [BlaSparseCSR.c](#).

#### 9.30.2.2 fasp\_dcsr\_bandwidth()

```
INT fasp_dcsr_bandwidth (
    const dCSRmat * A )
```

Get bandwith of matrix.

## Parameters

<i>A</i>	pointer to the <a href="#">dCSRmat</a> matrix
----------	---

## Author

Zheng Li

## Date

03/22/2015

Definition at line 242 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 1074 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

## Note

This routine can be modified for filtering.

Definition at line 1154 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 840 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 38 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 670 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 <code>dCSRmat</code> matrix
----------	--

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 170 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
<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 441 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 ( $0 \leq n \leq A.col-1$ )
<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 591 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 527 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 1382 of file BlaSparseCSR.c.

**9.30.2.13 fasp\_dcsr\_null()**

```
void fasp_dcsr_null (
    dCSRmat * A )
```

Initialize CSR sparse matrix.

## Parameters

<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix
----------	---

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 208 of file BlaSparseCSR.c.

## 9.30.2.14 fasp\_dcsr\_perm()

```
dCSRmat fasp_dcsr_perm (
    dCSRmat * A,
    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="#">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 272 of file BlaSparseCSR.c.

## 9.30.2.15 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: `Aperm=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 1603 of file `BlaSparseCSR.c`.

## 9.30.2.16 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 <code>diag(A)</code> 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 776 of file BlaSparseCSR.c.

**9.30.2.17 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 1202 of file BlaSparseCSR.c.

**9.30.2.18 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 383 of file `BlaSparseCSR.c`.

**9.30.2.19 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 1635 of file `BlaSparseCSR.c`.

## 9.30.2.20 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 1263 of file BlaSparseCSR.c.

## 9.30.2.21 fasp\_dcsr\_sympart()

```
dCSRmat fasp_dcsr_sympart (
    dCSRmat * A )
```

Get symmetric part of a [dCSRmat](#) matrix.

## Parameters

<i>*A</i>	pointer to the <a href="#">dCSRmat</a> matrix
-----------	---

## Returns

symmetrized the [dCSRmat](#) matrix

## Author

Xiaozhe Hu

## Date

03/21/2011

Definition at line 1349 of file BlaSparseCSR.c.

## 9.30.2.22 fasp\_dcsr\_trans()

```
void fasp_dcsr_trans (
    const dCSRmat * A,
    dCSRmat * AT )
```

Find transpose of **dCSRmat** matrix A.

## Parameters

A	Pointer to the <b>dCSRmat</b> matrix
AT	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 943 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

A	Pointer to matrix in <b>dCSRmat</b> for transpose, INPUT
p	Permutation, INPUT
AT	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  $p = \text{NULL}$  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  $A = \text{NULL}$ , then only transposes/permutes the structure of  $A$ .

For  $p = \text{NULL}$ , applying this two times  $A \rightarrow AT \rightarrow A$  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 1483 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 815 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 84 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

Definition at line 189 of file BlaSparseCSR.c.

## 9.30.2.27 fasp\_icsr\_null()

```
void fasp_icsr_null (
    iCSRmat * A )
```

Initialize CSR sparse matrix.

## Parameters

<i>A</i>	Pointer to the <i>iCSRmat</i> matrix
----------	--------------------------------------

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 225 of file BlaSparseCSR.c.

## 9.30.2.28 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 <i>iCSRmat</i> matrix <i>A</i>
<i>AT</i>	Pointer to the <i>iCSRmat</i> matrix <i>A'</i>

**Returns**

The transpose of [iCSRmat](#) matrix A

**Author**

Chensong Zhang

**Date**

04/06/2010

Modified by Chunsheng Feng, Zheng Li on 06/20/2012

Definition at line 867 of file [BlaSparseCSR.c](#).

## 9.31 [BlaSparseCSRL.c](#) File Reference

Sparse matrix operations for [dCSRLmat](#) matrices.

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

**Functions**

- [dCSRLmat](#) \* [fasp\\_dcsrl\\_create](#) (const [INT](#) num\_rows, const [INT](#) num\_cols, const [INT](#) num\_nonzeros)  
*Create a [dCSRLmat](#) object.*
- void [fasp\\_dcsrl\\_free](#) ([dCSRLmat](#) \*A)  
*Destroy a [dCSRLmat](#) object.*

### 9.31.1 Detailed Description

Sparse matrix operations for [dCSRLmat](#) matrices.

**Note**

This file contains Level-1 (Bla) functions. It requires [AuxMemory.c](#)  
Refer to John Mellor-Crummey and John Garvin Optimizaing sparse matrix vector product computations using unroll and jam, Tech Report Rice Univ, Aug 2002.

### 9.31.2 Function Documentation

#### 9.31.2.1 [fasp\\_dcsrl\\_create\(\)](#)

```
dCSRLmat * fasp\_dcsrl\_create (  
    const INT num_rows,  
    const INT num_cols,  
    const INT num_nonzeros )
```

Create a [dCSRLmat](#) 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

01/07/2001

Definition at line 33 of file BlaSparseCSRL.c.

## 9.31.2.2 fasp\_dcsrl\_free()

```
void fasp_dcsrl_free (
    dCSRLmat * A )
```

Destroy a [dCSRLmat](#) object.

## Parameters

<i>A</i>	Pointer to the <a href="#">dCSRLmat</a> type matrix
----------	---

## Author

Zhiyang Zhou

## Date

01/07/2011

Definition at line 61 of file BlaSparseCSRL.c.

## 9.32 BlaSparseSTR.c File Reference

Sparse matrix operations for [dSTRmat](#) matrices.

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

## Functions

- void `fasp_dstr_null` (`dSTRmat *A`)  
*Initialize sparse matrix on structured grid.*
- `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](#)

### 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

<code>nx</code>	Number of grids in x direction
<code>ny</code>	Number of grids in y direction
<code>nz</code>	Number of grids in z direction
<code>nxy</code>	Number of grids in x-y plane
<code>ngrid</code>	Number of grids
<code>nband</code>	Number of off-diagonal bands
<code>nc</code>	Number of components
<code>offsets</code>	Shift from diagonal

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

05/17/2010

Definition at line 112 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 <a href="#">dSTRmat</a> matrix
<i>B</i>	Pointer to the <a href="#">dSTRmat</a> matrix

**Author**

Zhiyang Zhou

**Date**

04/21/2010

Definition at line 184 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 60 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**

<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix
----------	---

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

05/17/2010

Definition at line 155 of file BlaSparseSTR.c.

## 9.32.2.5 fasp\_dstr\_null()

```
void fasp_dstr_null (
    dSTRmat * A )
```

Initialize sparse matrix on structured grid.

## Parameters

<i>A</i>	Pointer to the <code>dSTRmat</code> matrix
----------	--

## Author

Shiquan Zhang, Xiaozhe Hu

## Date

05/17/2010

Definition at line 28 of file BlasSparseSTR.c.

## 9.33 BlasSparseUtil.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: calculating the numerical values in the result.*
- 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: calculating the numerical values in the result.*
- 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  $R*A*P$ , 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 \*nyp, 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  $R*A*P$  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 independet 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

C-version: by Ludmil Zikatanov 2010-04-08 tested 2010-04-08

: Modified Xiaozhe Hu 2010-10-18

#### Note

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

### 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 276 of file BlasSparseUtil.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: calculating the numerical values in the result.

## 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 128 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>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 57 of file BlasSparseUtil.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
<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 363 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: calculating the numerical values in the result.

#### 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 435 of file BlaSparseUtil.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

**Note**

For the concrete algorithm, see:

Definition at line 201 of file BlasSparseUtil.c.

**9.33.2.7 fasp\_sparse\_MIS()**

```
ivector fasp_sparse_MIS (
    dCSRmat * A )
```

get the maximal independet set of a CSR matrix

**Parameters**

<b>A</b>	pointer to the matrix
----------	-----------------------

**Note**

: only use the sparsity of A, index starts from 1 (fortran)!!

Definition at line 912 of file BlasSparseUtil.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 792 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 non zeroes.

## 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>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 518 of file BlasSparseUtil.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
<i>v</i>	:O: the result $v^t = w^t A$
<i>nvp</i>	:I: number of nonzeros in $v$

Definition at line 652 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 $w[jw]$ is nonzero
<i>ia</i>	:I: array of row pointers for $A$
<i>ja</i>	:I: array of column indices for $A$
<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
<i>nvp</i>	:I: number of nonzeros in $v$
<i>icp</i>	:IO: is a working array of length $(*map)$ which on output satisfies $icp[jv[k]-1]=k$ ; Values of $icp[]$ at positions $*$ other than $(jv[k]-1)$ remain unchanged.

Modified by Chensong Zhang on 09/11/2012

Definition at line 599 of file BlasSparseUtil.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 nonzeros 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 nonzeros in x
<i>icp</i>	???
<i>s</i>	:O: $s = y^t x$ .

Definition at line 737 of file BlasSparseUtil.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>	:l: indices such that $y[jy]$ is nonzero
<i>y</i>	:l: is a sparse vector.
<i>nyp</i>	:l: number of nonzeros in $v$
<i>x</i>	:l: 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 703 of file `BlaSparseUtil.c`.

## 9.34 BlaSpmvBLC.c File Reference

BLAS operations for `dBLCmat` matrices.

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

### Functions

- void `fasp_blas_dbldc_aApy` (const `REAL` *alpha*, const `dBLCmat` \**A*, const `REAL` \**x*, `REAL` \**y*)  
*Matrix-vector multiplication  $y = \alpha * A * x + y$ .*
- void `fasp_blas_dbldc_mxv` (const `dBLCmat` \**A*, const `REAL` \**x*, `REAL` \**y*)  
*Matrix-vector multiplication  $y = A * x$ .*

### 9.34.1 Detailed Description

BLAS operations for `dBLCmat` matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires `BlaSpmvCSR.c`

### 9.34.2 Function Documentation

#### 9.34.2.1 `fasp_blas_dbldc_aApy()`

```
void fasp_blas_dbldc_aApy (
    const REAL alpha,
    const dBLCmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = \alpha * A * x + y$ .



## Parameters

<i>alpha</i>	REAL factor a
<i>A</i>	Pointer to <a href="#">dBLCmat</a> 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 33 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 A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

## Author

Chensong Zhang

## Date

04/27/2013

Definition at line 157 of file BlasSpmvBLC.c.

## 9.35 BlaSpmvBSR.c File Reference

BLAS 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

BLAS operations for [dBSRmat](#) matrices.

#### Note

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

### 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 dBSRmat 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 62 of file BlasSpmvBSR.c.

## 9.35.2.2 fasp\_blas\_dbsr\_aAcpy()

```
void fasp_blas_dbsr_aAcpy (
    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 343 of file `BlaSpmvBSR.c`.**9.35.2.3 fasp\_blas\_dbsr\_aApy\_agg()**

```
void fasp_blas_dbsr_aApy_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 619 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 33 of file BlasSpmvBSR.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 4641 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 905 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 dBSRmat 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 2692 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 <a href="#">dBSRmat</a> matrix
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>P</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>B</i>	Pointer to <a href="#">dBSRmat</a> 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 4951 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 <a href="#">dBSRmat</a> matrix
<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>P</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>B</i>	Pointer to <a href="#">dBSRmat</a> 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 4762 of file BlasSpmvBSR.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 5216 of file BlasSpmvBSR.c.

## 9.36 BlasSpmvCSR.c File Reference

BLAS 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  $B = R * A * P$ .*

### 9.36.1 Detailed Description

BLAS operations for **dCSRmat** matrices.

#### Note

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

## 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 482 of file BlaSpmvCSR.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 597 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 52 of file BlasSpmvCSR.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 204 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 763 of file `BlaSpmvCSR.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 228 of file BlasSpmvCSR.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 dCSRmat 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 425 of file BlasSpmvCSR.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 1603 of file BlaSpmvCSR.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 871 of file BlasPmvCSR.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 1703 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  $B=R*A*P$ .

## 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 $R*A*P$
<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 1801 of file BlaSpmvCSR.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 $R \cdot A \cdot P$

## Author

Xiaozhe Hu

## Date

05/10/2010

Modified by Chunsheng Feng, Xiaoqiang Yue on 05/26/2012

Definition at line 1151 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  $B=R \cdot A \cdot P$  (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 $R \cdot A \cdot P$

## 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 1417 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 708 of file BlaspmvCSR.c.

**9.37 BlaspmvCSRL.c File Reference**

BLAS 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

BLAS operations for [dCSRmat](#) matrices.

#### Note

This file contains Level-1 (Bla) functions.

Refer to John Mellor-Crummey and John Garvin Optimizaing sparse matrix vector product computations using unroll and jam, Tech Report Rice Univ, Aug 2002.

### 9.37.2 Function Documentation

#### 9.37.2.1 fasp\_blas\_dcsr\_mvv()

```
void fasp_blas_dcsr_mvv (
    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 <a href="#">dCSRmat</a> 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 30 of file BlasSpmvCSR.c.

## 9.38 BlasSpmvSTR.c File Reference

BLAS 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_mxv` (const `dSTRmat` \*A, const `REAL` \*x, `REAL` \*y)  
*Matrix-vector multiplication  $y = A * x$ .*
- `INT` `fasp_dstr_diagscale` (const `dSTRmat` \*A, `dSTRmat` \*B)  
 *$B = D^{-1}A$ .*

### 9.38.1 Detailed Description

BLAS operations for `dSTRmat` matrices.

#### Note

This file contains Level-1 (Bla) functions. It requires `AuxArray.c`, `AuxMemory.c`, `BlaSmallMatInv.c`, `BlaSmallMat.c`, and `BlaSparseSTR.c`

### 9.38.2 Function Documentation

#### 9.38.2.1 `fasp_blas_dstr_aAxy()`

```
void fasp_blas_dstr_aAxy (
    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 <code>dSTRmat</code> 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 56 of file BlasSpmvSTR.c.

## 9.38.2.2 fasp\_blas\_dstr\_mxv()

```
void fasp_blas_dstr_mxv (
    const dSTRmat * A,
    const REAL * x,
    REAL * y )
```

Matrix-vector multiplication  $y = A*x$ .

## Parameters

<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

Chensong Zhang

## Date

04/27/2013

Definition at line 126 of file BlasSpmvSTR.c.

## 9.38.2.3 fasp\_dstr\_diagscale()

```
INT fasp_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

**Date**

08/30/2012

Definition at line 151 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_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 error of two dvector x and y.*
- `REAL` `fasp_blas_dvec_norm1` (const `dvector` \*x)  
*L1 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.*



### 9.39.1 Detailed Description

BLAS1 operations for vectors.

#### Note

This file contains Level-1 (Bla) functions. It requires [AuxMessage.c](#) and [BlaArray.c](#)

### 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 )
```

$y = a*x + y$

#### 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 36 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**

<i>a</i>	REAL factor a
<i>x</i>	Pointer to dvector x
<i>y</i>	Pointer to dvector y
<i>z</i>	Pointer to dvector z

**Author**

Chensong Zhang

**Date**

07/01/209

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

Definition at line 88 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**

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

**Returns**

Inner product

**Author**

Chensong Zhang

**Date**

07/01/209

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

Definition at line 123 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

##### Date

05/23/2012

Definition at line 220 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

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

Definition at line 262 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\_inf norm of x

**Author**

Chensong Zhang

**Date**

07/01/2009

Definition at line 302 of file BlaVector.c.

## 9.39.2.7 fasp\_blas\_dvec\_relerr()

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

Relative error of two dvector x and y.

**Parameters**

$x$	Pointer to dvector $x$
$y$	Pointer to dvector $y$

**Returns**

relative error  $||x-y||/||x||$

**Author**

Chensong Zhang

**Date**

07/01/209

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

Definition at line 165 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.

## 9.41 fasp.h File Reference

Main header file for FASP.

```
#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 (or IJ) format.*
- struct [iCOOmat](#)  
*Sparse matrix of INT type in COO (or 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 [ILU\\_param](#)  
*Parameters for ILU.*
- struct [ILU\\_data](#)  
*Data for ILU setup.*
- struct [Schwarz\\_param](#)  
*Parameters for Schwarz method.*
- struct [Mumps\\_data](#)  
*Parameters for MUMPS interface.*
- struct [Pardiso\\_data](#)  
*Parameters for Intel MKL PARDISO interface.*
- struct [Schwarz\\_data](#)  
*Data for Schwarz methods.*
- struct [AMG\\_param](#)  
*Parameters for AMG solver.*
- struct [AMG\\_data](#)  
*Data for AMG solvers.*
- struct [precond\\_data](#)  
*Data passed to the preconditioners.*
- struct [precond\\_data\\_str](#)  
*Data passed to the preconditioner for [dSTRmat](#) matrices.*
- struct [precond\\_diagstr](#)  
*Data passed to diagonal preconditioner for [dSTRmat](#) matrices.*
- struct [precond](#)  
*Preconditioner data and action.*
- struct [mxv\\_matfree](#)

*Matrix-vector multiplication, replace the actual matrix.*

- struct [input\\_param](#)

*Input parameters.*

- struct [itsolver\\_param](#)

*Parameters passed to iterative solvers.*

## Macros

- #define [\\_\\_FASP\\_HEADER\\_\\_](#)

- #define [FASP\\_VERSION](#) 1.9

*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 [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))

- #define [FASP\\_GSRB](#) 1

## 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](#)



## Variables

- unsigned INT `total_alloc_mem`
- unsigned INT `total_alloc_count`  
*Total allocated memory amount.*
- INT `nx_rb`
- INT `ny_rb`
- INT `nz_rb`
- INT \* `IMAP`
- INT `MAXIMAP`
- INT `count`

### 9.41.1 Detailed Description

Main header file for FASP.

This header file contains general constants and data structures for FASP.

#### Note

Only define macros and data structures, no function declarations.

Created by Chensong Zhang on 08/12/2010. Modified by Chensong Zhang on 12/13/2011. Modified by Chensong Zhang on 12/25/2011. Modified by Chensong Zhang on 01/25/2015: clean up code Modified by Chensong Zhang on 01/27/2015: remove N2C, C2N, ISTART Modified by Ludmil Zikatanov on 20151011: cosmetics.

Modified by Hongxuan Zhang on 11/28/2015: add Intel MKL PARDISO support.

### 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 36 of file `fasp.h`.

#### 9.41.2.2 `ABS`

```
#define ABS(  
    a ) ((a)>=0.0)?(a):-(a)
```

absolute value of a

Definition at line 77 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 61 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 50 of file fasp.h.

#### 9.41.2.5 FASP\_GSRB

```
#define FASP_GSRB 1
```

MG level 0 use RedBlack Gauss Seidel Smoothing

Definition at line 1162 of file fasp.h.

#### 9.41.2.6 FASP\_VERSION

```
#define FASP_VERSION 1.9
```

FASP base version information.

fapsolver version

Definition at line 45 of file fasp.h.

#### 9.41.2.7 GE

```
#define GE(  
    a,  
    b )  ( ( (a) >= (b) ) ? (TRUE) : (FALSE) )
```

is  $a \geq b$ ?

Definition at line 83 of file fasp.h.

#### 9.41.2.8 GT

```
#define GT(  
    a,  
    b )  ( ( (a) > (b) ) ? (TRUE) : (FALSE) )
```

Definition of  $>$ ,  $\geq$ ,  $<$ ,  $\leq$ , and `isnan`.

is  $a > b$ ?

Definition at line 82 of file fasp.h.

#### 9.41.2.9 INT

```
#define INT int
```

regular integer type: int or long

Definition at line 67 of file fasp.h.

#### 9.41.2.10 ISNAN

```
#define ISNAN(  
    a )  ( ( (a) != (a) ) ? (TRUE) : (FALSE) )
```

is  $a == \text{NAN}$ ?

Definition at line 86 of file fasp.h.

#### 9.41.2.11 LE

```
#define LE(  
    a,  
    b )  ((a)<=(b))?(TRUE):(FALSE)
```

is  $a \leq b$ ?

Definition at line 85 of file fasp.h.

#### 9.41.2.12 LONG

```
#define LONG long
```

long integer type

Definition at line 68 of file fasp.h.

#### 9.41.2.13 LONGLONG

```
#define LONGLONG long long
```

long integer type

Definition at line 69 of file fasp.h.

#### 9.41.2.14 LS

```
#define LS(  
    a,  
    b )  ((a)<(b))?(TRUE):(FALSE)
```

is  $a < b$ ?

Definition at line 84 of file fasp.h.

#### 9.41.2.15 MAX

```
#define MAX(  
    a,  
    b )  ( ( (a) > (b) ) ? (a) : (b) )
```

Definition of max, min, abs.

bigger one in a and b

Definition at line 75 of file fasp.h.

#### 9.41.2.16 MIN

```
#define MIN(  
    a,  
    b )  ( ( (a) < (b) ) ? (a) : (b) )
```

smaller one in a and b

Definition at line 76 of file fasp.h.

#### 9.41.2.17 NEDMALLOC

```
#define NEDMALLOC OFF
```

use nedmalloc instead of standard malloc

Definition at line 51 of file fasp.h.

#### 9.41.2.18 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 91 of file fasp.h.

#### 9.41.2.19 PUT\_REAL

```
#define PUT_REAL(  
    A ) printf("### DEBUG: %s = %e\n", #A, (A))
```

print real num

Definition at line 92 of file fasp.h.

#### 9.41.2.20 REAL

```
#define REAL double
```

float type

Definition at line 70 of file fasp.h.

#### 9.41.2.21 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 59 of file fasp.h.

#### 9.41.2.22 SHORT

```
#define SHORT short
```

FASP integer and floating point numbers.

short integer type

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

### 9.41.4 Variable Documentation

#### 9.41.4.1 count

```
INT count
```

Counter for multiple calls

#### 9.41.4.2 IMAP

```
INT* IMAP
```

Red Black Gs Smoother imap



#### 9.41.4.3 MAXIMAP

`INT` MAXIMAP

Red Black Gs Smoother max DOFs of reservoir

#### 9.41.4.4 nx\_rb

`INT` nx\_rb

Red Black Gs Smoother Nx

#### 9.41.4.5 ny\_rb

`INT` ny\_rb

Red Black Gs Smoother Ny

#### 9.41.4.6 nz\_rb

`INT` nz\_rb

Red Black Gs Smoother Nz

#### 9.41.4.7 total\_alloc\_count

`unsigned INT` total\_alloc\_count

Total allocated memory amount.

total allocation times

Definition at line 39 of file AuxMemory.c.

#### 9.41.4.8 total\_alloc\_mem

`unsigned INT` total\_alloc\_mem

total allocated memory

Definition at line 38 of file AuxMemory.c.

## 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\\_ivec](#)  
*Block INT vector structure.*
- struct [AMG\\_data\\_bsr](#)  
*Data for multigrid levels. (BSR format)*
- struct [precond\\_diagbsr](#)  
*Data passed to diagonal preconditioner for [dBSRmat](#) matrices.*
- struct [precond\\_data\\_bsr](#)  
*Data passed to the preconditioners.*
- struct [precond\\_block\\_data](#)  
*Data passed to the preconditioner for block preconditioning for [dBLCmat](#) format.*
- struct [precond\\_sweeping\\_data](#)  
*Data passed to the preconditioner for sweeping preconditioning.*

### 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\\_ivec](#) [block\\_ivec](#)

### 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.

Created by Chensong Zhang on 05/21/2010. Modified by Xiaozhe Hu on 05/28/2010: add preconditioner↔\_reservoir\_data. Modified by Xiaozhe Hu on 06/15/2010: modify preconditioner\_block\_reservoir\_data. Modified by Chensong Zhang on 10/11/2010: add BSR data. Modified by Chensong Zhang on 10/17/2012: modify comments. Modified by Ludmil Zikatanov on 10/11/2015: cosmetics.

Modified by Chensong Zhang on 01/13/2017: remove reservoir simulation part

### 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 23 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 dBSRmat

```
typedef struct dBSRmat dBSRmat
```

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 all kinds of messages, including error messages, solver types, etc.

#### Macros

- #define [FASP\\_SUCCESS](#) 0  
*Definition of return status and error messages.*
- #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\\_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\\_VBiCGstab](#) 9
  - #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\\_SBiCGstab](#) 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 V\_CYCLE 1

*Definition of cycle types.*

- #define W\_CYCLE 2
- #define AMLI\_CYCLE 3
- #define NL\_AMLI\_CYCLE 4
- #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 [ILU\\_MC\\_OMP](#) 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 all kinds of messages, including error messages, solver types, etc.

#### Note

This is internal use only. Do NOT change.

Created by Chensong Zhang on 03/20/2010. Modified by Chensong Zhang on 12/06/2011. Modified by Chensong Zhang on 12/25/2011. Modified by Chensong Zhang on 04/22/2012. Modified by Ludmil Zikatanov on 02/15/2013: CG -> SMOOTHER\_CG. Modified by Chensong Zhang on 02/16/2013: GS -> SMOOTHER\_GS, etc. Modified by Chensong Zhang on 04/09/2013: Add safe Krylov methods. Modified by Chensong Zhang on 09/22/2013: Clean up Doxygen.

Modified by Chensong Zhang on 09/17/2013: Filename changed from message.h.

### 9.43.2 Macro Definition Documentation

#### 9.43.2.1 AMLI\_CYCLE

```
#define AMLI_CYCLE 3
```

AMLI-cycle

Definition at line 184 of file fasp\_const.h.

#### 9.43.2.2 ASCEND

```
#define ASCEND 12
```

Ascending order

Definition at line 246 of file fasp\_const.h.

#### 9.43.2.3 BIGREAL

```
#define BIGREAL 1e+20
```

Some global constants.

A large real number

Definition at line 252 of file fasp\_const.h.



#### 9.43.2.4 CF\_ORDER

```
#define CF_ORDER 1
```

C/F order smoothing

Definition at line 237 of file fasp\_const.h.

#### 9.43.2.5 CGPT

```
#define CGPT 1
```

Coarse grid points

Definition at line 230 of file fasp\_const.h.

#### 9.43.2.6 CLASSIC\_AMG

```
#define CLASSIC_AMG 1
```

Definition of AMG types.

classic AMG

Definition at line 169 of file fasp\_const.h.

#### 9.43.2.7 COARSE\_AC

```
#define COARSE_AC 4
```

Aggressive coarsening

Definition at line 213 of file fasp\_const.h.

#### 9.43.2.8 COARSE\_CR

```
#define COARSE_CR 3
```

Compatible relaxation

Definition at line 212 of file fasp\_const.h.

#### 9.43.2.9 COARSE\_MIS

```
#define COARSE_MIS 5
```

Aggressive coarsening based on MIS

Definition at line 214 of file fasp\_const.h.

#### 9.43.2.10 COARSE\_RS

```
#define COARSE_RS 1
```

Definition of coarsening types.

Classical

Definition at line 210 of file fasp\_const.h.

#### 9.43.2.11 COARSE\_RSP

```
#define COARSE_RSP 2
```

Classical, with positive offdiags

Definition at line 211 of file fasp\_const.h.

#### 9.43.2.12 CPFIRST

```
#define CPFIRST 1
```

C-points first order

Definition at line 244 of file fasp\_const.h.

#### 9.43.2.13 DESCEND

```
#define DESCEND 21
```

Descending order

Definition at line 247 of file fasp\_const.h.

#### 9.43.2.14 ERROR\_ALLOC\_MEM

```
#define ERROR_ALLOC_MEM -20
```

fail to allocate memory

Definition at line 37 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 44 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 45 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 42 of file fasp\_const.h.

#### 9.43.2.18 ERROR\_AMG\_SMOOTH\_TYPE

```
#define ERROR_AMG_SMOOTH_TYPE -31
```

unknown smoother type

Definition at line 43 of file fasp\_const.h.

**9.43.2.19 ERROR\_DATA\_STRUCTURE**

```
#define ERROR_DATA_STRUCTURE -21
```

problem with data structures

Definition at line 38 of file fasp\_const.h.

**9.43.2.20 ERROR\_DATA\_ZERODIAG**

```
#define ERROR_DATA_ZERODIAG -22
```

matrix has zero diagonal entries

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

**9.43.2.21 ERROR\_DUMMY\_VAR**

```
#define ERROR_DUMMY_VAR -23
```

unexpected input data

Definition at line 40 of file fasp\_const.h.

**9.43.2.22 ERROR\_INPUT\_PAR**

```
#define ERROR_INPUT_PAR -13
```

wrong input argument

Definition at line 31 of file fasp\_const.h.

**9.43.2.23 ERROR\_LIC\_TYPE**

```
#define ERROR_LIC_TYPE -80
```

wrong license type

Definition at line 60 of file fasp\_const.h.

#### 9.43.2.24 ERROR\_MAT\_SIZE

```
#define ERROR_MAT_SIZE -15
```

wrong problem size

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

#### 9.43.2.25 ERROR\_MISC

```
#define ERROR_MISC -19
```

other error

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

#### 9.43.2.26 ERROR\_NUM\_BLOCKS

```
#define ERROR_NUM_BLOCKS -18
```

wrong number of blocks

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

#### 9.43.2.27 ERROR\_OPEN\_FILE

```
#define ERROR_OPEN_FILE -10
```

fail to open a file

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

#### 9.43.2.28 ERROR\_QUAD\_DIM

```
#define ERROR_QUAD_DIM -61
```

unsupported quadrature dim

Definition at line 58 of file fasp\_const.h.

**9.43.2.29 ERROR\_QUAD\_TYPE**

```
#define ERROR_QUAD_TYPE -60
```

unknown quadrature type

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

**9.43.2.30 ERROR\_REGRESS**

```
#define ERROR_REGRESS -14
```

regression test fail

Definition at line 32 of file fasp\_const.h.

**9.43.2.31 ERROR\_SOLVER\_EXIT**

```
#define ERROR_SOLVER_EXIT -49
```

solver does not quit successfully

Definition at line 55 of file fasp\_const.h.

**9.43.2.32 ERROR\_SOLVER\_ILUSETUP**

```
#define ERROR_SOLVER_ILUSETUP -45
```

ILU setup error

Definition at line 52 of file fasp\_const.h.

**9.43.2.33 ERROR\_SOLVER\_MAXIT**

```
#define ERROR_SOLVER_MAXIT -48
```

maximal iteration number exceeded

Definition at line 54 of file fasp\_const.h.

#### 9.43.2.34 ERROR\_SOLVER\_MISC

```
#define ERROR_SOLVER_MISC -46
```

misc solver error during run time

Definition at line 53 of file fasp\_const.h.

#### 9.43.2.35 ERROR\_SOLVER\_PRECTYPE

```
#define ERROR_SOLVER_PRECTYPE -41
```

unknown precondition type

Definition at line 48 of file fasp\_const.h.

#### 9.43.2.36 ERROR\_SOLVER\_SOLSTAG

```
#define ERROR_SOLVER_SOLSTAG -43
```

solver's solution is too small

Definition at line 50 of file fasp\_const.h.

#### 9.43.2.37 ERROR\_SOLVER\_STAG

```
#define ERROR_SOLVER_STAG -42
```

solver stagnates

Definition at line 49 of file fasp\_const.h.

#### 9.43.2.38 ERROR\_SOLVER\_TOLSMALL

```
#define ERROR_SOLVER_TOLSMALL -44
```

solver's tolerance is too small

Definition at line 51 of file fasp\_const.h.

**9.43.2.39 ERROR\_SOLVER\_TYPE**

```
#define ERROR_SOLVER_TYPE -40
```

unknown solver type

Definition at line 47 of file fasp\_const.h.

**9.43.2.40 ERROR\_UNKNOWN**

```
#define ERROR_UNKNOWN -99
```

an unknown error type

Definition at line 62 of file fasp\_const.h.

**9.43.2.41 ERROR\_WRONG\_FILE**

```
#define ERROR_WRONG_FILE -11
```

input contains wrong format

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

**9.43.2.42 FALSE**

```
#define FALSE 0
```

logic FALSE

Definition at line 68 of file fasp\_const.h.

**9.43.2.43 FASP\_SUCCESS**

```
#define FASP_SUCCESS 0
```

Definition of return status and error messages.

return from function successfully

Definition at line 27 of file fasp\_const.h.



#### 9.43.2.44 FGPT

```
#define FGPT 0
```

Fine grid points

Definition at line 229 of file fasp\_const.h.

#### 9.43.2.45 FPFIRST

```
#define FPFIRST -1
```

F-points first order

Definition at line 245 of file fasp\_const.h.

#### 9.43.2.46 G0PT

```
#define G0PT -5
```

Type of vertices (DOFs) for coarsening.

Cannot fit in aggregates

Definition at line 227 of file fasp\_const.h.

#### 9.43.2.47 ILU\_MC\_OMP

```
#define ILU_MC_OMP 1
```

Multi-colors Parallel smoothing

Definition at line 238 of file fasp\_const.h.

**9.43.2.48 ILUk**

```
#define ILUk 1
```

Type of ILU methods.

ILUk

Definition at line 155 of file fasp\_const.h.

**9.43.2.49 ILUt**

```
#define ILUt 2
```

ILUt

Definition at line 156 of file fasp\_const.h.

**9.43.2.50 ILUtp**

```
#define ILUtp 3
```

ILUtp

Definition at line 157 of file fasp\_const.h.

**9.43.2.51 INTERP\_DIR**

```
#define INTERP_DIR 1
```

Definition of interpolation types.

Direct interpolation

Definition at line 219 of file fasp\_const.h.

#### 9.43.2.52 INTERP\_ENG

```
#define INTERP_ENG 3
```

Energy minimization interpolation

Definition at line 221 of file fasp\_const.h.

#### 9.43.2.53 INTERP\_EXT

```
#define INTERP_EXT 6
```

Extended interpolation

Definition at line 222 of file fasp\_const.h.

#### 9.43.2.54 INTERP\_STD

```
#define INTERP_STD 2
```

Standard interpolation

Definition at line 220 of file fasp\_const.h.

#### 9.43.2.55 ISPT

```
#define ISPT 2
```

Isolated points

Definition at line 231 of file fasp\_const.h.

#### 9.43.2.56 MAT\_bBSR

```
#define MAT_bBSR 12
```

block BSR/CSR matrix

Definition at line 101 of file fasp\_const.h.

**9.43.2.57 MAT\_bCSR**

```
#define MAT_bCSR 11
```

block CSR/CSR matrix == 2\*2 BLC matrix

Definition at line 100 of file fasp\_const.h.

**9.43.2.58 MAT\_BLC**

```
#define MAT_BLC 8
```

block CSR matrix

Definition at line 96 of file fasp\_const.h.

**9.43.2.59 MAT\_BSR**

```
#define MAT_BSR 2
```

block-wise compressed sparse row

Definition at line 92 of file fasp\_const.h.

**9.43.2.60 MAT\_bSTR**

```
#define MAT_bSTR 13
```

block STR/CSR matrix

Definition at line 102 of file fasp\_const.h.

**9.43.2.61 MAT\_CSR**

```
#define MAT_CSR 1
```

compressed sparse row

Definition at line 91 of file fasp\_const.h.

#### 9.43.2.62 MAT\_CSRL

```
#define MAT_CSRL 6
```

modified CSR to reduce cache missing

Definition at line 94 of file fasp\_const.h.

#### 9.43.2.63 MAT\_FREE

```
#define MAT_FREE 0
```

Definition of matrix format.

matrix-free format: only mxv action

Definition at line 89 of file fasp\_const.h.

#### 9.43.2.64 MAT\_STR

```
#define MAT_STR 3
```

structured sparse matrix

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

#### 9.43.2.65 MAT\_SymCSR

```
#define MAT_SymCSR 7
```

symmetric CSR format

Definition at line 95 of file fasp\_const.h.

#### 9.43.2.66 MAX\_AMG\_LVL

```
#define MAX_AMG_LVL 20
```

Maximal AMG coarsening level

Definition at line 256 of file fasp\_const.h.

**9.43.2.67 MAX\_CRATE**

```
#define MAX_CRATE 20.0
```

Maximal coarsening ratio

Definition at line 259 of file fasp\_const.h.

**9.43.2.68 MAX\_REFINE\_LVL**

```
#define MAX_REFINE_LVL 20
```

Maximal refinement level

Definition at line 255 of file fasp\_const.h.

**9.43.2.69 MAX\_RESTART**

```
#define MAX_RESTART 20
```

Maximal restarting number

Definition at line 260 of file fasp\_const.h.

**9.43.2.70 MAX\_STAG**

```
#define MAX_STAG 20
```

Maximal number of stagnation times

Definition at line 261 of file fasp\_const.h.

**9.43.2.71 MIN\_CDOF**

```
#define MIN_CDOF 20
```

Minimal number of coarsest variables

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

#### 9.43.2.72 MIN\_CRATE

```
#define MIN_CRATE 0.9
```

Minimal coarsening ratio

Definition at line 258 of file fasp\_const.h.

#### 9.43.2.73 NL\_AMLI\_CYCLE

```
#define NL_AMLI_CYCLE 4
```

Nonlinear AMLI-cycle

Definition at line 185 of file fasp\_const.h.

#### 9.43.2.74 NO\_ORDER

```
#define NO_ORDER 0
```

Definition of smoothing order.

Natural order smoothing

Definition at line 236 of file fasp\_const.h.

#### 9.43.2.75 OFF

```
#define OFF 0
```

turn off certain parameter

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

**9.43.2.76 ON**

```
#define ON 1
```

Definition of switch.

turn on certain parameter

Definition at line 73 of file fasp\_const.h.

**9.43.2.77 OPENMP\_HOLDS**

```
#define OPENMP_HOLDS 2000
```

Smallest size for OpenMP version

Definition at line 263 of file fasp\_const.h.

**9.43.2.78 PAIRWISE**

```
#define PAIRWISE 1
```

Definition of aggregation types.

pairwise aggregation

Definition at line 176 of file fasp\_const.h.

**9.43.2.79 PREC\_AMG**

```
#define PREC_AMG 2
```

with AMG precondition

Definition at line 147 of file fasp\_const.h.



**9.43.2.80 PREC\_DIAG**

```
#define PREC_DIAG 1
```

with diagonal precondition

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

**9.43.2.81 PREC\_FMG**

```
#define PREC_FMG 3
```

with full AMG precondition

Definition at line 148 of file fasp\_const.h.

**9.43.2.82 PREC\_ILU**

```
#define PREC_ILU 4
```

with ILU precondition

Definition at line 149 of file fasp\_const.h.

**9.43.2.83 PREC\_NULL**

```
#define PREC_NULL 0
```

Definition of preconditioner type for iterative methods.

with no precondition

Definition at line 145 of file fasp\_const.h.

**9.43.2.84 PREC\_SCHWARZ**

```
#define PREC_SCHWARZ 5
```

with Schwarz preconditioner

Definition at line 150 of file fasp\_const.h.

**9.43.2.85 PRINT\_ALL**

```
#define PRINT_ALL 10
```

all: all printouts, including files

Definition at line 84 of file fasp\_const.h.

**9.43.2.86 PRINT\_MIN**

```
#define PRINT_MIN 1
```

quiet: print error, important warnings

Definition at line 80 of file fasp\_const.h.

**9.43.2.87 PRINT\_MORE**

```
#define PRINT_MORE 4
```

more: print some useful debug info

Definition at line 82 of file fasp\_const.h.

**9.43.2.88 PRINT\_MOST**

```
#define PRINT_MOST 8
```

most: maximal printouts, no files

Definition at line 83 of file fasp\_const.h.

**9.43.2.89 PRINT\_NONE**

```
#define PRINT_NONE 0
```

Print level for all subroutines – not including DEBUG output.

silent: no printout at all

Definition at line 79 of file fasp\_const.h.

#### 9.43.2.90 PRINT\_SOME

```
#define PRINT_SOME 2
```

some: print less important warnings

Definition at line 81 of file fasp\_const.h.

#### 9.43.2.91 SA\_AMG

```
#define SA_AMG 2
```

smoothed aggregation AMG

Definition at line 170 of file fasp\_const.h.

#### 9.43.2.92 SCHWARZ\_BACKWARD

```
#define SCHWARZ_BACKWARD 2
```

Backward ordering

Definition at line 163 of file fasp\_const.h.

#### 9.43.2.93 SCHWARZ\_FORWARD

```
#define SCHWARZ_FORWARD 1
```

Type of Schwarz smoother.

Forward ordering

Definition at line 162 of file fasp\_const.h.

#### 9.43.2.94 SCHWARZ\_SYMMETRIC

```
#define SCHWARZ_SYMMETRIC 3
```

Symmetric smoother

Definition at line 164 of file fasp\_const.h.

**9.43.2.95 SMALLREAL**

```
#define SMALLREAL 1e-20
```

A small real number

Definition at line 253 of file fasp\_const.h.

**9.43.2.96 SMALLREAL2**

```
#define SMALLREAL2 1e-40
```

An extremely small real number

Definition at line 254 of file fasp\_const.h.

**9.43.2.97 SMOOTHER\_BLKOil**

```
#define SMOOTHER_BLKOil 11
```

Definition of specialized smoother types.

Used in monolithic AMG for black-oil

Definition at line 204 of file fasp\_const.h.

**9.43.2.98 SMOOTHER\_CG**

```
#define SMOOTHER_CG 4
```

CG as a smoother

Definition at line 193 of file fasp\_const.h.

**9.43.2.99 SMOOTHER\_GS**

```
#define SMOOTHER_GS 2
```

Gauss-Seidel smoother

Definition at line 191 of file fasp\_const.h.

#### 9.43.2.100 SMOOTHER\_GSOR

```
#define SMOOTHER_GSOR 7
```

GS + SOR smoother

Definition at line 196 of file fasp\_const.h.

#### 9.43.2.101 SMOOTHER\_JACOBI

```
#define SMOOTHER_JACOBI 1
```

Definition of standard smoother types.

Jacobi smoother

Definition at line 190 of file fasp\_const.h.

#### 9.43.2.102 SMOOTHER\_L1DIAG

```
#define SMOOTHER_L1DIAG 10
```

L1 norm diagonal scaling smoother

Definition at line 199 of file fasp\_const.h.

#### 9.43.2.103 SMOOTHER\_POLY

```
#define SMOOTHER_POLY 9
```

Polynomial smoother

Definition at line 198 of file fasp\_const.h.

#### 9.43.2.104 SMOOTHER\_SGS

```
#define SMOOTHER_SGS 3
```

Symmetric Gauss-Seidel smoother

Definition at line 192 of file fasp\_const.h.

**9.43.2.105 SMOOTHER\_SGSOR**

```
#define SMOOTHER_SGSOR 8
```

SGS + SSOR smoother

Definition at line 197 of file fasp\_const.h.

**9.43.2.106 SMOOTHER\_SOR**

```
#define SMOOTHER_SOR 5
```

SOR smoother

Definition at line 194 of file fasp\_const.h.

**9.43.2.107 SMOOTHER\_SPETEN**

```
#define SMOOTHER_SPETEN 19
```

Used in monolithic AMG for black-oil

Definition at line 205 of file fasp\_const.h.

**9.43.2.108 SMOOTHER\_SSOR**

```
#define SMOOTHER_SSOR 6
```

SSOR smoother

Definition at line 195 of file fasp\_const.h.

**9.43.2.109 SOLVER\_AMG**

```
#define SOLVER_AMG 21
```

AMG as an iterative solver

Definition at line 127 of file fasp\_const.h.

**9.43.2.110 SOLVER\_BiCGstab**

```
#define SOLVER_BiCGstab 2
```

Bi-Conjugate Gradient Stabilized

Definition at line 110 of file fasp\_const.h.

**9.43.2.111 SOLVER\_CG**

```
#define SOLVER_CG 1
```

Conjugate Gradient

Definition at line 109 of file fasp\_const.h.

**9.43.2.112 SOLVER\_DEFAULT**

```
#define SOLVER_DEFAULT 0
```

Definition of solver types for iterative methods.

Use default solver in FASP

Definition at line 107 of file fasp\_const.h.

**9.43.2.113 SOLVER\_FMG**

```
#define SOLVER_FMG 22
```

Full AMG as an solver

Definition at line 128 of file fasp\_const.h.

**9.43.2.114 SOLVER\_GCG**

```
#define SOLVER_GCG 7
```

Generalized Conjugate Gradient

Definition at line 116 of file fasp\_const.h.

**9.43.2.115 SOLVER\_GCR**

```
#define SOLVER_GCR 8
```

Generalized Conjugate Residual

Definition at line 117 of file fasp\_const.h.

**9.43.2.116 SOLVER\_GMRES**

```
#define SOLVER_GMRES 4
```

Generalized Minimal Residual

Definition at line 113 of file fasp\_const.h.

**9.43.2.117 SOLVER\_MinRes**

```
#define SOLVER_MinRes 3
```

Minimal Residual

Definition at line 112 of file fasp\_const.h.

**9.43.2.118 SOLVER\_MUMPS**

```
#define SOLVER_MUMPS 33
```

Direct Solver: MUMPS

Definition at line 132 of file fasp\_const.h.

**9.43.2.119 SOLVER\_PARDISO**

```
#define SOLVER_PARDISO 34
```

Direct Solver: PARDISO

Definition at line 133 of file fasp\_const.h.



**9.43.2.120 SOLVER\_SBiCGstab**

```
#define SOLVER_SBiCGstab 12
```

BiCGstab with safety net

Definition at line 120 of file fasp\_const.h.

**9.43.2.121 SOLVER\_SCG**

```
#define SOLVER_SCG 11
```

Conjugate Gradient with safety net

Definition at line 119 of file fasp\_const.h.

**9.43.2.122 SOLVER\_SGCG**

```
#define SOLVER_SGCG 17
```

GCG with safety net

Definition at line 125 of file fasp\_const.h.

**9.43.2.123 SOLVER\_SGMRES**

```
#define SOLVER_SGMRES 14
```

GMRes with safety net

Definition at line 122 of file fasp\_const.h.

**9.43.2.124 SOLVER\_SMinRes**

```
#define SOLVER_SMinRes 13
```

MinRes with safety net

Definition at line 121 of file fasp\_const.h.

**9.43.2.125 SOLVER\_SUPERLU**

```
#define SOLVER_SUPERLU 31
```

Direct Solver: SuperLU

Definition at line 130 of file fasp\_const.h.

**9.43.2.126 SOLVER\_SVFGMRES**

```
#define SOLVER_SVFGMRES 16
```

Variable-restart FGMRES with safety net

Definition at line 124 of file fasp\_const.h.

**9.43.2.127 SOLVER\_SVGMRES**

```
#define SOLVER_SVGMRES 15
```

Variable-restart GMRES with safety net

Definition at line 123 of file fasp\_const.h.

**9.43.2.128 SOLVER\_UMFPACK**

```
#define SOLVER_UMFPACK 32
```

Direct Solver: UMFPack

Definition at line 131 of file fasp\_const.h.

**9.43.2.129 SOLVER\_VBiCGstab**

```
#define SOLVER_VBiCGstab 9
```

VBi-Conjugate Gradient Stabilized

Definition at line 111 of file fasp\_const.h.

**9.43.2.130 SOLVER\_VFGMRES**

```
#define SOLVER_VFGMRES 6
```

Variable Restarting Flexible GMRES

Definition at line 115 of file fasp\_const.h.

**9.43.2.131 SOLVER\_VGMRES**

```
#define SOLVER_VGMRES 5
```

Variable Restarting GMRES

Definition at line 114 of file fasp\_const.h.

**9.43.2.132 STAG\_RATIO**

```
#define STAG_RATIO 1e-4
```

Stagnation tolerance =  $\text{tol} * \text{STAGRATIO}$

Definition at line 262 of file fasp\_const.h.

**9.43.2.133 STOP\_MOD\_REL\_RES**

```
#define STOP_MOD_REL_RES 3
```

modified relative residual  $\|r\|/\|x\|$

Definition at line 140 of file fasp\_const.h.

**9.43.2.134 STOP\_REL\_PRECRES**

```
#define STOP_REL_PRECRES 2
```

relative B-residual  $\|r\|_B/\|b\|_B$

Definition at line 139 of file fasp\_const.h.

**9.43.2.135 STOP\_REL\_RES**

```
#define STOP_REL_RES 1
```

Definition of iterative solver stopping criteria types.

relative residual  $\|r\|/\|b\|$

Definition at line 138 of file fasp\_const.h.

**9.43.2.136 TRUE**

```
#define TRUE 1
```

Definition of logic type.

logic TRUE

Definition at line 67 of file fasp\_const.h.

**9.43.2.137 UA\_AMG**

```
#define UA_AMG 3
```

unsmoothed aggregation AMG

Definition at line 171 of file fasp\_const.h.

**9.43.2.138 UNPT**

```
#define UNPT -1
```

Undetermined points

Definition at line 228 of file fasp\_const.h.

#### 9.43.2.139 USERDEFINED

```
#define USERDEFINED 0
```

Type of ordering for smoothers.

User defined order

Definition at line 243 of file fasp\_const.h.

#### 9.43.2.140 V\_CYCLE

```
#define V_CYCLE 1
```

Definition of cycle types.

V-cycle

Definition at line 182 of file fasp\_const.h.

#### 9.43.2.141 VMB

```
#define VMB 2
```

VMB aggregation

Definition at line 177 of file fasp\_const.h.

#### 9.43.2.142 W\_CYCLE

```
#define W_CYCLE 2
```

W-cycle

Definition at line 183 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.

Created by Chensong Zhang on 01/21/2017.

### 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 10 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 43 of file fasp\_grid.h.

## 9.44.3.3 pgrid2d

```
typedef grid2d* pgrid2d
```

Grid in 2d

Definition at line 41 of file fasp\_grid.h.

## 9.45 ltrSmootherBSR.c File Reference

Smoothers for dBSRmat matrices.

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

## Functions

- void [fasp\\_smoother\\_dbsr\\_jacobi](#) (dBSRmat \*A, dvector \*b, dvector \*u)  
*Jacobi relaxation.*
- void [fasp\\_smoother\\_dbsr\\_jacobi\\_setup](#) (dBSRmat \*A, dvector \*b, dvector \*u, 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_omp` = 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`, `AuxTiming.c`, `Bla↔SmallMatInv.c`, `BlaSmallMat.c`, `BlaArray.c`, `BlaSpmvBSR.c`, and `PreBSR.c`

## 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

<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>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 423 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 <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>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 542 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 A

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Definition at line 579 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

## 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 652 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 * diaginv )
```

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 721 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\_descend1' 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 795 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 <i>A</i>
<i>mark</i>	Pointer to the user-defined ordering

## Author

Zhiyang Zhou

## Date

2010/10/25

Definition at line 865 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 943 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 1559 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 46 of file ltrSmootherBSR.c.

## 9.45.2.11 fasp\_smoother\_dbsr\_jacobi1()

```
void fasp_smoother_dbsr_jacobi1 (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginvs )
```

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 269 of file ltrSmootherBSR.c.

#### 9.45.2.12 fasp\_smoother\_dbsr\_jacobi\_setup()

```
void fasp_smoother_dbsr_jacobi_setup (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    REAL * diaginvs )
```

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>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<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 161 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 <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>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 1020 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 * diaginvs,
    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 1142 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 1183 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 * diaginv,
    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 1307 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 1433 of file ltrSmootherBSR.c.

### 9.45.3 Variable Documentation

#### 9.45.3.1 ilu\_solve\_omp

`REAL ilu_solve_omp = 0.0`

ILU time for the SOLVE phase

Definition at line 26 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)  
*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.*
- void `fasp_smoother_dcsr_gs_rb3d` (`dvector` \*u, `dCSRmat` \*A, `dvector` \*b, `INT` L, const `INT` order, `INT` \*mark, const `INT` maximap, const `INT` nx, const `INT` ny, const `INT` nz)  
*Colored Gauss-Seidel smoother for Au=b.*

### 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](#), [BlaArray.c](#), and [BlaSpmvCSR.c](#)

### 9.46.2 Function Documentation

#### 9.46.2.1 fasp\_smoother\_dcsr\_gs()

```
void fasp_smoother_dcsr_gs (
    dvector * u,
    const INT i_l,
    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> <sub>←</sub> <i>1</i>	Starting index
<i>i</i> <sub>←</sub> <i>_</i> <sub>←</sub> <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

#### Author

Xuehai Huang, Chensong Zhang

## Date

09/26/2009

Modified by Chunsheng Feng, Zheng Li on 09/01/2012

Definition at line 198 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 367 of file ltrSmootherCSR.c.

## 9.46.2.3 fasp\_smoother\_dcsr\_gs\_rb3d()

```
void fasp_smoother_dcsr_gs_rb3d (
    dvector * u,
    dCSRmat * A,
    dvector * b,
    INT L,
    const INT order,
    INT * mark,
    const INT maximap,
    const INT nx,
    const INT ny,
    const INT nz )
```

Colored Gauss-Seidel smoother for  $Au=b$ .

## Parameters

<i>u</i>	Initial guess and the new approximation to the solution
<i>A</i>	Pointer to stiffness matrix
<i>b</i>	Pointer to right hand side
<i>L</i>	Number of iterations
<i>order</i>	Ordering: -1: Forward; 1: Backward
<i>mark</i>	Marker for C/F points
<i>maximap</i>	Size of IMAP
<i>nx</i>	Number vertex of X direction
<i>ny</i>	Number vertex of Y direction
<i>nz</i>	Number vertex of Z direction

## Author

Chunsheng Feng

## Date

02/08/2012

Definition at line 1429 of file ltrSmootherCSR.c.

## 9.46.2.4 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 1070 of file ltrSmootherCSR.c.

## 9.46.2.5 fasp\_smoother\_dcsr\_jacobi()

```
void fasp_smoother_dcsr_jacobi (
    dvector * u,
    const INT i_l,
    const INT i_n,
    const INT s,
    dCSRmat * A,
    dvector * b,
    INT L )
```

Jacobi method as a smoother.

## Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$i_{\leftarrow}$ $\_ \leftarrow$ <i>1</i>	Starting index
$i_{\leftarrow}$ $\_ \leftarrow$ <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



**Author**

Xuehai Huang, Chensong Zhang

**Date**

09/26/2009

Modified by Chunsheng Feng, Zheng Li on 08/29/2012

Definition at line 62 of file ltrSmootherCSR.c.

**9.46.2.6 fasp\_smoother\_dcsr\_kaczmarz()**

```

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.

**Parameters**

$u$	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$i_{\leftarrow 1}$	Starting index
$i_{\leftarrow n}$	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**

Xiaozhe Hu

**Date**

2010/11/12

Modified by Chunsheng Feng, Zheng Li on 2012/09/01

Definition at line 1149 of file ltrSmootherCSR.c.

**9.46.2.7 fasp\_smoother\_dcsr\_L1diag()**

```

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.

**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 <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 1290 of file ltrSmootherCSR.c.

## 9.46.2.8 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

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<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

## Author

Xiaozhe Hu

## Date

10/26/2010

Modified by Chunsheng Feng, Zheng Li on 09/01/2012

Definition at line 632 of file ltrSmootherCSR.c.

## 9.46.2.9 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)
----------	--

## Parameters

$i_{\leftarrow}$ $_{\leftarrow}$ 1	Starting index
$i_{\leftarrow}$ $_{\leftarrow}$ 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
w	Over-relaxation weight

## Author

Xiaozhe Hu

## Date

10/26/2010

Modified by Chunsheng Feng, Zheng Li on 09/01/2012

Definition at line 748 of file ltrSmootherCSR.c.

## 9.46.2.10 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

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
w	Over-relaxation weight
mark	C/F marker array
order	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 876 of file ltrSmootherCSR.c.

## 9.47 ltrSmootherCSRcr.c File Reference

Smoother 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

Smoother 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](#)

### 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 41 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"
#include "ItrAuxiliary.inl"
```

## 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](#), [BlaArray.c](#), and [BlaSpmvCSR.c](#). 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.

### 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 57 of file ltrSmootherCSRpoly.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 157 of file ltrSmootherCSRpoly.c.

## 9.49 ltrSmootherSTR.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\\_schwarz](#) (dSTRmat \*A, dvector \*b, dvector \*u, dvector \*diaginv, ivector \*pivot, ivector \*neigh, ivector \*order)  
*Schwarz method as the smoother.*

### 9.49.1 Detailed Description

Smoothers for [dSTRmat](#) matrices.

#### Note

This file contains Level-2 (ltr) functions. It requires [AuxArray.c](#), [AuxMemory.c](#), [AuxVector.c](#), [BlaSmallMat.c](#), [BlaSmallMatInv.c](#), [BlaSmallMatLU.c](#), and [BlaSpmvSTR.c](#)

## 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 1531 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 212 of file ltrSmootherSTR.c.

## 9.49.2.3 fasp\_smoother\_dstr\_gsl()

```
void fasp_smoother_dstr_gsl (
    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 272 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 * diaginvs )
```

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 317 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 672 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
<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 432 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 * diaginvs,
    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 549 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 39 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 <i>A</i> when $(A->nc)>1$ , and NULL when $(A->nc)=1$

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 88 of file ltrSmootherSTR.c.



## 9.49.2.10 fasp\_smoother\_dstr\_schwarz()

```
void fasp_smoother_dstr_schwarz (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    dvector * diaginv,
    ivector * pivot,
    ivector * neigh,
    ivector * order )
```

Schwarz 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>diaginv</i>	Pointer to dvector: the inverse of the diagonals
<i>pivot</i>	Pointer to ivector: the pivot of diagonal blocks
<i>neigh</i>	Pointer to ivector: neighborhoods
<i>order</i>	Pointer to ivector: the smoothing order

## Author

Xiaozhe Hu

## Date

10/01/2011

Definition at line 1653 of file ltrSmootherSTR.c.

## 9.49.2.11 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 864 of file ltrSmootherSTR.c.

## 9.49.2.12 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 926 of file ltrSmootherSTR.c.

**9.49.2.13 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 972 of file ltrSmootherSTR.c.

#### 9.49.2.14 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
<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 1344 of file ltrSmootherSTR.c.

#### 9.49.2.15 fasp\_smoother\_dstr\_sor\_descend()

```
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.

## 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 1092 of file ltrSmootherSTR.c.

## 9.49.2.16 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 1213 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](#) stop\_type, const [SHORT](#) prtlvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dbsr\\_pbcgs](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop\_type, const [SHORT](#) prtlvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dblc\\_pbcgs](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop\_type, const [SHORT](#) prtlvl)  
*A preconditioned BiCGstab method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dstr\\_pbcgs](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop\_type, const [SHORT](#) prtlvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_pbcgs](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop\_type, 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](#)

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

See [KrySPbcgs.c](#) for a safer version

## 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 stop_type,
    const SHORT prtlvl )

```

A preconditioned BiCGstab method for solving  $Au=b$ .

#### 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>stop_type</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

Rewritten by Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 06/01/2012: fix restart param-init Modified by Chensong Zhang on 03/31/2013

Definition at line 740 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ .

#### 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>stop_type</i>	Stopping criteria type
<i>prtlvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

09/09/2009

Rewritten by Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 06/01/2012: fix restart param-init Modified by Chensong Zhang on 03/31/2013

Definition at line 398 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ .

## 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>stop_type</i>	Stopping criteria type
<i>prtlvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

09/09/2009

Rewritten by Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 06/01/2012: fix restart param-init Modified by Chensong Zhang on 03/31/2013

Definition at line 56 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ .

##### 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>stop_type</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

Rewritten by Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 06/01/2012: fix restart param-init Modified by Chensong Zhang on 03/31/2013

Definition at line 1082 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 stop_type,
    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: 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>stop_type</i>	Stopping criteria type
<i>prtlvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Chensong Zhang

## Date

09/09/2009

Rewritten by Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 06/01/2012: fix restart param-init Modified by Feiteng Huang on 09/26/2012, (mmatrix free)

Definition at line 1424 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, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` stop\_type, const `SHORT` prtlvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- `INT fasp_solver_dbsr_pcg` (`dBsrmat` \*A, `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` stop\_type, const `SHORT` prtlvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- `INT fasp_solver_dbldc_pcg` (`dBLCmat` \*A, `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` stop\_type, const `SHORT` prtlvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- `INT fasp_solver_dstr_pcg` (`dSTRmat` \*A, `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` stop\_type, const `SHORT` prtlvl)  
*Preconditioned conjugate gradient method for solving  $Au=b$ .*
- `INT fasp_solver_pcg` (`mxv_matfree` \*mf, `dvector` \*b, `dvector` \*u, `precond` \*pc, const `REAL` tol, const `INT` MaxIt, const `SHORT` stop\_type, const `SHORT` prtlvl)  
*Preconditioned conjugate gradient (CG) method for solving  $Au=b$ .*

### 9.51.1 Detailed Description

Krylov subspace methods – Preconditioned CG.

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:  $r_{k+1} = r_k - \alpha(Ap_k)$ ;
- 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

#### 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](#).  
Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [KrySPcg.c](#) for a safer version

## 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned conjugate gradient 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>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>stop_type</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 04/30/2012 Modified by Chensong Zhang on 03/28/2013

Definition at line 675 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 stop_type,
    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>stop_type</i>	Stopping criteria type

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

05/26/2014

Definition at line 383 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 stop_type,  
    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>stop_type</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 Chensong Zhang on 04/30/2012 Modified by Chensong Zhang on 03/28/2013

Definition at line 94 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 stop_type,
    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>stop_type</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 04/30/2012 Modified by Chensong Zhang on 03/28/2013

Definition at line 967 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 stop_type,
    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>stop_type</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 Chensong Zhang on 04/30/2012 Modified by Feiteng Huang on 09/19/2012: matrix free

Definition at line 1259 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](#) stop\_type, 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](#) stop\_type, 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](#)

Refer to 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

### 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 stop_type,
    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>stop_type</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 52 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 stop_type,
    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>stop_type</i>	Stopping criteria type – Not implemented

**Returns**

Iteration number if converges; ERROR otherwise.

**Author**

Xiaozhe Hu

**Date**

01/01/2012

**Note**

Not completely implemented yet! –Chensong

Modified by Chensong Zhang on 05/01/2012 Modified by Feiteng Huang on 09/26/2012: matrix free

Definition at line 203 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 stop_type, 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](#)

## 9.53.2 Function Documentation

### 9.53.2.1 fasp\_solver\_dcsr\_pgcr()

```

INT fasp_solver_dcsr_pgcr (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT stop_type,
    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>stop_type</i>	Stopping type
<i>prtlvl</i>	How much information to print out

#### Returns

Iteration number if converges; ERROR otherwise.

#### Note

Refer to YVAN NOTAY "AN AGGREGATION-BASED ALGEBRAIC MULTIGRID METHOD"

#### Author

Zheng Li

#### Date

12/23/2014

Definition at line 46 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, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) stop\_type, const [SHORT](#) prtIvl)  
*Right 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](#) stop\_type, const [SHORT](#) prtIvl)  
*Preconditioned GMRES method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_dbsr\\_pgmres](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) restart, const [SHORT](#) stop\_type, const [SHORT](#) prtIvl)  
*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](#) stop\_type, const [SHORT](#) prtIvl)  
*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](#) stop\_type, const [SHORT](#) prtIvl)  
*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](#)

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

See also [KryPvgmres.c](#) for a variable restarting version.

See [KrySPgmres.c](#) for a safer version

### 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned GMRES method for solving  $Au=b$ .

## 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 precondition: structure of precondition
<i>tol</i>	Tolerance for stopping
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restarting steps
<i>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/05/2013: add stop\_type and safe check

Definition at line 363 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 stop_type,
    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>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/05/2013: add stop\_type and safe check

Definition at line 666 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 stop_type,
    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>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/05/2013: Add stop\_type and safe check Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate Modified by Chensong Zhang on 07/30/2014: Make memory allocation size long int Modified by Chensong Zhang on 09/21/2014: Add comments and reorganize code

Definition at line 60 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 stop_type,
    const SHORT prtlvl )

```

Preconditioned GMRES 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>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>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/05/2013: add stop\_type and safe check

Definition at line 970 of file KryPgmres.c.

## 9.54.2.5 fasp\_solver\_pgmres()

```

INT fasp_solver_pgmres (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT stop_type,
    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>stop_type</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 Chensong Zhang on 05/01/2012 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 1274 of file KryPgmres.c.

## 9.55 KryPminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual.

```
#include <math.h>
#include "fasp.h"
#include "fasp_funcs.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](#) stop\_type, 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](#) stop\_type, 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](#) stop\_type, 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](#) stop\_type, 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](#).  
Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [KrySPminres.c](#) for a safer version

### 9.55.2 Function Documentation

#### 9.55.2.1 fasp\_solver\_dblc\_pminres()

```
INT fasp_solver_dblc_pminres (
    dBLCmat * A,
    dvector * b,
    dvector * u,
    precond * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT stop_type,
    const SHORT prtlvl )
```

A preconditioned minimal residual (Minres) 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>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>stop_type</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

## Note

Rewritten based on the original version by Xiaozhe Hu 05/24/2010

Modified by Chensong Zhang on 04/09/2013

Definition at line 462 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 stop_type,
    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>stop_type</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

## Note

Rewritten based on the original version by Shiquan Zhang 05/10/2010

Modified by Chensong Zhang on 04/09/2013

Definition at line 55 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 stop_type,
    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>stop_type</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 865 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 stop_type,
    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>stop_type</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 Modified by Feiteng Huang on 09/26/2012: matrix free

Definition at line 1271 of file KryPminres.c.

## 9.56 KryPvbcgs.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\\_pvbcgs](#) (dCSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.*
- [INT fasp\\_solver\\_dbsr\\_pvbcgs](#) (dBSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.*
- [INT fasp\\_solver\\_dblc\\_pvbcgs](#) (dBLCmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.*
- [INT fasp\\_solver\\_dstr\\_pvbcgs](#) (dSTRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.*
- [INT fasp\\_solver\\_pvbcgs](#) (mxv\_matfree \*mf, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.*



### 9.56.1 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](#). Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM. See [KrySPbcgs.c](#) for a safer version.

### 9.56.2 Function Documentation

#### 9.56.2.1 fasp\_solver\_dblc\_pvbcgs()

```

INT fasp_solver_dblc_pvbcgs (
    dBLMat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT stop_type,
    const SHORT prtlvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.

#### 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>stop_type</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 KryPvbcgs.c.

**9.56.2.2 fasp\_solver\_dbsr\_pvbcgs()**

```
INT fasp_solver_dbsr_pvbcgs (
    dBSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT stop_type,
    const SHORT prtlvl )
```

Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.**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>stop_type</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 384 of file KryPvbcgs.c.

## 9.56.2.3 fasp\_solver\_dcsr\_pvbcgs()

```
INT fasp_solver_dcsr_pvbcgs (
    dCSRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT stop_type,
    const SHORT prtlvl )
```

Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.

## 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>stop_type</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 53 of file KryPvbcgs.c.

#### 9.56.2.4 fasp\_solver\_dstr\_pvbcs()

```

INT fasp_solver_dstr_pvbcs (
    dSTRmat * A,
    dvector * b,
    dvector * u,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT stop_type,
    const SHORT prtlvl )

```

Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.

##### 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>stop_type</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 1046 of file KryPvbcs.c.

#### 9.56.2.5 fasp\_solver\_pvbcs()

```

INT fasp_solver_pvbcs (
    mxv_matfree * mf,
    dvector * b,
    dvector * u,

```

```
precond * pc,  
const REAL tol,  
const INT MaxIt,  
const SHORT stop_type,  
const SHORT prtlvl )
```

Preconditioned BiCGstab method for solving  $Au=b$ , Rewritten from Matlab 2011a.

## 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>stop_type</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 1377 of file KryPvbcgs.c.

## 9.57 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, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop\_type, 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, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop\_type, 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](#) (dBLCmat \*A, dvector \*b, dvector \*x, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop\_type, 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, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop\_type, 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.*

### 9.57.1 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](#)

Refer to A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMR(m) Journal of Computational and Applied Mathematics, 230 (2009) pp. 751-761. UCRL-JRNL-235266.

This file is modified from [KryPvfgmres.c](#)

### 9.57.2 Function Documentation

#### 9.57.2.1 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 stop_type,
    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>stop_type</i>	default stopping criterion, i.e. $\ r_k\ /\ r_0\  < \text{tol}$ , is used.
<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 718 of file KryPvfgmres.c.

**9.57.2.2 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 stop_type,
    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>stop_type</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**

02/05/2012

Modified by Chensong Zhang on 05/01/2012 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 388 of file KryPvfgmres.c.

**9.57.2.3 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 stop_type,
    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>stop_type</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**

01/04/2012

Modified by Chensong Zhang on 05/01/2012 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 60 of file KryPvfgmres.c.

**9.57.2.4 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 stop_type,
    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 <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>stop_type</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**

01/04/2012

Modified by Chensong Zhang on 05/01/2012 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 1046 of file KryPvgrimres.c.

## 9.58 KryPvgrimres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRES.

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

**Functions**

- `INT fasp_solver_dcsr_pvgrimres` (`dCSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT stop_type`, `const SHORT prtlvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dblc_pvgrimres` (`dBLCmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT stop_type`, `const SHORT prtlvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dbsr_pvgrimres` (`DBSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT stop_type`, `const SHORT prtlvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_dstr_pvgrimres` (`dSTRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT restart`, `const SHORT stop_type`, `const SHORT prtlvl`)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during iteration.*
- `INT fasp_solver_pvgrimres` (`mxv_matfree *mf`, `dvector *b`, `dvector *x`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `SHORT restart`, `const SHORT stop_type`, `const SHORT prtlvl`)  
*Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during iteration.*

### 9.58.1 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](#)

Refer to 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.

See [KrySPvgmres.c](#) for a safer version

### 9.58.2 Function Documentation

#### 9.58.2.1 fasp\_solver\_dblc\_pvgmres()

```
INT fasp_solver_dblc_pvgmres (
    dBLMat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    const SHORT restart,
    const SHORT stop_type,
    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>stop_type</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 402 of file KryPvgmres.c.

## 9.58.2.2 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 stop_type,
    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>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/06/2013: Add stop type support

Definition at line 747 of file KryPvgmres.c.

## 9.58.2.3 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 stop_type,
    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>stop_type</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 12/13/2011 Modified by Chensong Zhang on 05/01/2012 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 60 of file KryPvgmres.c.

## 9.58.2.4 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 stop_type,
    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>stop_type</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 05/01/2012 Modified by Chensong Zhang on 04/06/2013: Add stop type support

Definition at line 1092 of file KryPvgmres.c.

## 9.58.2.5 fasp\_solver\_pvgmres()

```

INT fasp_solver_pvgmres (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    precondition * pc,
    const REAL tol,
    const INT MaxIt,
    SHORT restart,
    const SHORT stop_type,
    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 <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>stop_type</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 Chensong Zhang on 12/13/2011 Modified by Chensong Zhang on 05/01/2012 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 1439 of file KryPvbmres.c.

## 9.59 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\\_spbcgs](#) (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\\_spbcgs](#) (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\\_spbcgs](#) (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.*

### 9.59.1 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](#). Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM See [KryPbcgs.c](#) for a version without safety net

## 9.59.2 Function Documentation

### 9.59.2.1 fasp\_solver\_dblc\_spgbcs()

```

INT fasp_solver_dblc_spgbcs (
    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 <b>dBLCmat</b> : 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 827 of file KrySPbcs.c.

## 9.59.2.2 fasp\_solver\_dbsr\_spgcgs()

```

INT fasp_solver_dbsr_spgcgs (
    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 439 of file KrySPbcgs.c.

## 9.59.2.3 fasp\_solver\_dcsr\_spgcgs()

```

INT fasp_solver_dcsr_spgcgs (
    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.

## 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 51 of file KrySPbcgs.c.

## 9.59.2.4 fasp\_solver\_dstr\_spbcs()

```

INT fasp_solver_dstr_spbcs (
    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 1215 of file KrySPbcgs.c.

## 9.60 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.*

### 9.60.1 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](#). Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM. See [KryPcg.c](#) for a version without safety net.

## 9.60.2 Function Documentation

### 9.60.2.1 fasp\_solver\_dblc\_spcg()

```

INT fasp_solver_dblc_spcg (
    const dBLCmat * 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 <b>dBLCmat</b> : 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 381 of file KrySPcg.c.

### 9.60.2.2 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 51 of file KrySPcg.c.

### 9.60.2.3 fasp\_solver\_dstr\_spcg()

```

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.

## 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 711 of file KrySPcg.c.

## 9.61 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\\_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\\_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\\_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.*

### 9.61.1 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](#)

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

See also [pgmres.c](#) for a variable restarting version.

See [KryPgmres.c](#) for a version without safety net

### 9.61.2 Function Documentation

#### 9.61.2.1 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 397 of file KrySPgmres.c.

**9.61.2.2 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 738 of file KrySPgmres.c.

## 9.61.2.3 fasp\_solver\_dcsr\_spgmres()

```

INT fasp_solver_dcsr_spgmres (
    const dCSRmat * 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="#">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 56 of file KrySPgmres.c.

#### 9.61.2.4 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 <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 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 1079 of file KrySPgmres.c.

## 9.62 KrySPminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual 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.

### 9.62.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual 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](#)

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

See [KryPminres.c](#) for a version without safety net

### 9.62.2 Function Documentation

#### 9.62.2.1 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 <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 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 499 of file KrySPminres.c.

**9.62.2.2 fasp\_solver\_dcsr\_spminres()**

```

INT fasp_solver_dcsr_spminres (
    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 51 of file KrySPminres.c.

## 9.62.2.3 fasp\_solver\_dstr\_spminres()

```
INT fasp_solver_dstr_spminres (
    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 947 of file KrySPminres.c.

## 9.63 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\\_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\\_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\\_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.*

### 9.63.1 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](#)

Refer to 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.

See [KryPvgmres.c](#) a version without safety net

### 9.63.2 Function Documentation

## 9.63.2.1 fasp\_solver\_dbic\_spvgmres()

```

INT fasp_solver_dbic_spvgmres (
    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 dBLMat: 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 436 of file KrySPvgmres.c.

### 9.63.2.2 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 <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

04/06/2013

Definition at line 815 of file KrySPvgmres.c.

## 9.63.2.3 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 58 of file KrySPvgmres.c.

#### 9.63.2.4 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 <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 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 1194 of file KrySPvgmres.c.

## 9.64 PreAMGCoarsenCR.c File Reference

Coarsening with Brannick-Falgout strategy.

```

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

### 9.64.1 Detailed Description

Coarsening with Brannick-Falgout strategy.

#### Note

This file contains Level-4 (Pre) functions. It requires [AuxMemory.c](#) and [ltrSmootherCSRcr.c](#)

### 9.64.2 Function Documentation

#### 9.64.2.1 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 53 of file PreAMGCoarsenCR.c.

## 9.65 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.*

### 9.65.1 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](#), [AuxVector.c](#), [Blas](#), [SparseCSR.c](#), and [PreAMGCoarsenCR.c](#)

Refer to 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.

**Warning**

Do NOT use auto-indentation in this file!!!

### 9.65.2 Function Documentation



## 9.65.2.1 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 69 of file PreAMGCoarsenRS.c.

## 9.66 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.*
- void [fasp\\_amg\\_interp\\_trunc](#) ([dCSRmat](#) \*P, [AMG\\_param](#) \*param)  
*Truncation step for prolongation operators.*

### 9.66.1 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](#), and [PreAMG↔InterpEM.c](#)

Refer to 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.

### 9.66.2 Function Documentation

#### 9.66.2.1 [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
<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 55 of file PreAMGInterp.c.

**9.66.2.2 fasp\_amg\_interp\_trunc()**

```
void fasp_amg_interp_trunc (
    dCSRmat * P,
    AMG_param * param )
```

Truncation step for prolongation operators.

**Parameters**

<i>P</i>	Prolongation (input: full, output: truncated)
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

**Author**

Chensong Zhang

**Date**

05/14/2013

Originally by Xuehai Huang, Chensong Zhang on 01/31/2009 Modified by Chunsheng Feng, Xiaoqiang Yue on 05/23/2012: add OMP support Modified by Chensong Zhang on 05/14/2013: rewritten

Definition at line 110 of file PreAMGInterp.c.

**9.67 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.*

### 9.67.1 Detailed Description

Interpolation operators for AMG based on energy-min.

#### Note

This file contains Level-4 (Pre) functions. It requires [AuxArray.c](#), [AuxMemory.c](#), [AuxVector.c](#), [BlaSmallMatLU.c](#), [BlaSparseCSR.c](#), [KryPcg.c](#), and [PreCSR.c](#)

Refer to J. Xu and L. Zikatanov On An Energy Minimizing Basis in Algebraic Multigrid Methods Computing and visualization in sciences, 2003

### 9.67.2 Function Documentation

#### 9.67.2.1 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 57 of file PreAMGInterpEM.c.

## 9.68 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.*

### 9.68.1 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 [PreAMG↵CoarsenCR.c](#)

Setup A, P, R and levels using the Compatible Relaxation coarsening for classic AMG interpolation Refer to J. Brannick and R. Falgout Compatible relaxation and coarsening in AMG

#### Warning

Not working. Yet need to be fixed. –Chensong

### 9.68.2 Function Documentation

#### 9.68.2.1 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 41 of file PreAMGSetupCR.c.

## 9.69 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.*

### 9.69.1 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](#), [BlaILUSetupCSR.c](#), [BlaSchwarzSetup.c](#), [BlaSparseCSR.c](#), [BlaSpmvCSR.c](#), [PreAMGCoarsenRS.c](#), [PreAMGInterp.c](#), and [PreMGRecurAMLI.c](#)

Refer to 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.

## 9.69.2 Function Documentation

### 9.69.2.1 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 Chensong Zhang on 04/04/2009. Modified by Chensong Zhang on 05/09/2010. Modified by Zhiyang Zhou on 11/17/2010. Modified by Xiaozhe Hu on 01/23/2011: add AMLI cycle. Modified by Chensong zhang on 09/09/2011↔: add min dof. Modified by Xiaozhe Hu on 04/24/2013: aggressive coarsening. Modified by Chensong Zhang on 05/03/2013: add error handling in setup. Modified by Chensong Zhang on 05/10/2013: adjust the structure. Modified by Chensong Zhang on 07/26/2014: handle coarsening errors. Modified by Chensong Zhang on 09/23/2014: check coarse spaces.

Definition at line 52 of file PreAMGSetupRS.c.

## 9.70 PreAMGSetupSA.c File Reference

Smoothed aggregation AMG: SETUP phase.

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

## Functions

- [SHORT fasp\\_amg\\_setup\\_sa](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of smoothed aggregation AMG.*

### 9.70.1 Detailed Description

Smoothed 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; Refer to P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994

### 9.70.2 Function Documentation

#### 9.70.2.1 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 Chensong Zhang on 04/06/2010. Modified by Chensong Zhang on 05/09/2010. 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 57 of file PreAMGSetupSA.c.

## 9.71 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 "PreAMGAggregationBSR.inl"
#include "PreAMGAggregation.inl"
```

### Functions

- [SHORT fasp\\_amg\\_setup\\_sa\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of smoothed aggregation AMG (BSR format)*

#### 9.71.1 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; Refer to P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994

#### 9.71.2 Function Documentation

##### 9.71.2.1 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 51 of file PreAMGSetupSABSR.c.

## 9.72 PreAMGSetupUA.c File Reference

Unsmoothed aggregation AMG: SETUP phase.

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

**Functions**

- [SHORT fasp\\_amg\\_setup\\_ua](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of unsmoothed aggregation AMG.*

### 9.72.1 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; Refer to P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994

## 9.72.2 Function Documentation

### 9.72.2.1 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 47 of file PreAMGSetupUA.c.

## 9.73 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 "PreAMGAggregationBSR.inl"
#include "PreAMGAggregation.inl"
```

## Functions

- [SHORT fasp\\_amg\\_setup\\_ua\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of unsmoothed aggregation AMG (BSR format)*

### 9.73.1 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](#), [BlaILUSetupBSR.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; Refer to P. Vanek, J. Madel and M. Brezina Algebraic Multigrid on Unstructured Meshes, 1994

### 9.73.2 Function Documentation

#### 9.73.2.1 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 47 of file PreAMGSetupUABSR.c.

## 9.74 PreBLC.c File Reference

Preconditioners for [dBLMat](#) matrices.

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

### Functions

- void [fasp\\_precond\\_block\\_diag\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block diagonal preconditioning (3x3 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_diag\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block diagonal preconditioning (3x3 block matrix, each diagonal block is solved by AMG)*
- void [fasp\\_precond\\_block\\_diag\\_4](#) (REAL \*r, REAL \*z, void \*data)  
*block diagonal preconditioning (4x4 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_lower\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_lower\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved by AMG)*
- void [fasp\\_precond\\_block\\_lower\\_4](#) (REAL \*r, REAL \*z, void \*data)  
*block lower triangular preconditioning (4x4 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_upper\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_upper\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved AMG)*
- void [fasp\\_precond\\_block\\_SGS\\_3](#) (REAL \*r, REAL \*z, void \*data)  
*block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_block\\_SGS\\_3\\_amg](#) (REAL \*r, REAL \*z, void \*data)  
*block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)*
- void [fasp\\_precond\\_sweeping](#) (REAL \*r, REAL \*z, void \*data)  
*sweeping preconditioner for Maxwell equations*

### 9.74.1 Detailed Description

Preconditioners for [dBLMat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires [AuxArray.c](#), [AuxMemory.c](#), [AuxVector.c](#), [BlaSpmvCSR.c](#), and [PreMGCycle.c](#)

#### Warning

Need to be cleaned up. –Chensong

## 9.74.2 Function Documentation

### 9.74.2.1 fasp\_precond\_block\_diag\_3()

```
void fasp_precond_block_diag_3 (
    REAL * r,
    REAL * z,
    void * data )
```

block diagonal preconditioning (3x3 block matrix, each diagonal block is solved exactly)

#### 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

Definition at line 31 of file PreBLC.c.

### 9.74.2.2 fasp\_precond\_block\_diag\_3\_amg()

```
void fasp_precond_block_diag_3_amg (
    REAL * r,
    REAL * z,
    void * data )
```

block diagonal preconditioning (3x3 block matrix, each diagonal block is solved by AMG)

#### 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

Definition at line 115 of file PreBLC.c.

**9.74.2.3 fasp\_precond\_block\_diag\_4()**

```
void fasp_precond_block_diag_4 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block diagonal preconditioning (4x4 block matrix, each diagonal block is solved exactly)

**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

Definition at line 180 of file PreBLC.c.

**9.74.2.4 fasp\_precond\_block\_lower\_3()**

```
void fasp_precond_block_lower_3 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)

**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

Definition at line 276 of file PreBLC.c.

**9.74.2.5 fasp\_precond\_block\_lower\_3\_amg()**

```
void fasp_precond_block_lower_3_amg (
    REAL * r,
    REAL * z,
    void * data )
```

block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved by AMG)

**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

Definition at line 362 of file PreBLC.c.



#### 9.74.2.6 fasp\_precond\_block\_lower\_4()

```
void fasp_precond_block_lower_4 (
    REAL * r,
    REAL * z,
    void * data )
```

block lower triangular preconditioning (4x4 block matrix, each diagonal block is solved exactly)

##### 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

Definition at line 436 of file PreBLC.c.

#### 9.74.2.7 fasp\_precond\_block\_SGS\_3()

```
void fasp_precond_block_SGS_3 (
    REAL * r,
    REAL * z,
    void * data )
```

block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)

##### 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

Definition at line 704 of file PreBLC.c.

## 9.74.2.8 fasp\_precond\_block\_SGS\_3\_amg()

```
void fasp_precond_block_SGS_3_amg (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)

## 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

Definition at line 813 of file PreBLC.c.

## 9.74.2.9 fasp\_precond\_block\_upper\_3()

```
void fasp_precond_block_upper_3 (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)

## 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

Definition at line 538 of file PreBLC.c.

**9.74.2.10 fasp\_precond\_block\_upper\_3\_amg()**

```
void fasp_precond_block_upper_3_amg (  
    REAL * r,  
    REAL * z,  
    void * data )
```

block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved AMG)

**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

Definition at line 624 of file PreBLC.c.

**9.74.2.11 fasp\_precond\_sweeping()**

```
void fasp_precond_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

Definition at line 922 of file PreBLC.c.

## 9.75 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\\_nl\\_amli](#) (REAL \*r, REAL \*z, void \*data)  
*Nonlinear AMLI-cycle AMG preconditioner.*
- void [fasp\\_precond\\_dbsr\\_amg\\_nk](#) (REAL \*r, REAL \*z, void \*data)  
*AMG with extra near kernel solve preconditioner.*

### 9.75.1 Detailed Description

Preconditioners for [dBSRmat](#) matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires [AuxArray.c](#), [AuxParam.c](#), [AuxVector.c](#), [BlaSmallMat.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [KrySPcg.c](#), [KrySPvgmres.c](#), [PreMGCycle.c](#), and [PreMGRecurAMLI.c](#)

### 9.75.2 Function Documentation

#### 9.75.2.1 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 974 of file PreBSR.c.

#### 9.75.2.2 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 1053 of file PreBSR.c.

**9.75.2.3 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

**Date**

05/24/2012

**Note**

Works for general nb (Xiaozhe)

Definition at line 45 of file PreBSR.c.

**9.75.2.4 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

**Date**

05/24/2012

**Note**

Works for 2-component (Xiaozhe)

Definition at line 118 of file PreBSR.c.

**9.75.2.5 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

**Date**

05/24/2012

**Note**

Works for 3-component (Xiaozhe)

Definition at line 167 of file PreBSR.c.

**9.75.2.6 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

**Date**

05/24/2012

**Note**

Works for 5-component (Xiaozhe)

Definition at line 216 of file PreBSR.c.

**9.75.2.7 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

**Date**

05/24/2012

**Note**

Works for 7-component (Xiaozhe)

Definition at line 265 of file PreBSR.c.

**9.75.2.8 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.

**9.75.2.9 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

ZhengLi

## Date

12/04/2016

## Note

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

Definition at line 767 of file PreBSR.c.

## 9.75.2.10 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

ZhengLi

## Date

12/04/2016

**Note**

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

Definition at line 569 of file PreBSR.c.

**9.75.2.11 fasp\_precond\_dbsr\_nl\_amli()**

```
void fasp_precond_dbsr_nl_amli (
    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 1017 of file PreBSR.c.

**9.76 PreCSR.c File Reference**

Preconditioners for [dCSRmat](#) matrices.

```
#include "fasp.h"
#include "fasp_functs.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_schwarz` (`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_nl_amli` (`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.*
- void `fasp_precond_free` (const `SHORT` `precond_type`, `precond` \*`pc`)  
*free preconditioner*

### 9.76.1 Detailed Description

Preconditioners for `dCSRmat` matrices.

#### Note

This file contains Level-4 (Pre) functions. It requires `AuxArray.c`, `AuxMemory.c`, `AuxParam.c`, `AuxVector.c`, `BlaL←LUSetupCSR.c`, `BlaSchwarzSetup.c`, `BlaSparseCSR.c`, `BlaSpmvCSR.c`, `KrySPcg.c`, `KrySPvgmres.c`, `PreAMG←SetupRS.c`, `PreAMGSetupSA.c`, `PreAMGSetupUA.c`, `PreDataInit.c`, `PreMGCycle.c`, `PreMGCycleFull.c`, and `Pre←MGRecurAMLI.c`

### 9.76.2 Function Documentation

#### 9.76.2.1 `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 408 of file PreCSR.c.

**9.76.2.2 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 540 of file PreCSR.c.

### 9.76.2.3 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 474 of file PreCSR.c.

### 9.76.2.4 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 167 of file PreCSR.c.

**9.76.2.5 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 441 of file PreCSR.c.

**9.76.2.6 fasp\_precond\_free()**

```
void fasp_precond_free (
    const SHORT precond_type,
    precondition * pc )
```

free preconditioner

**Parameters**

<i>precond_type</i>	Preconditioner type
* <i>pc</i>	precondition data & fct



**Returns**

void

**Author**

Feiteng Huang

**Date**

12/24/2012

Definition at line 624 of file PreCSR.c.

**9.76.2.7 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 193 of file PreCSR.c.

#### 9.76.2.8 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 310 of file PreCSR.c.

**9.76.2.9 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, Shiquang Zhang

**Date**

04/06/2010

Definition at line 257 of file PreCSR.c.

#### 9.76.2.10 fasp\_precond\_nl\_amli()

```
void fasp_precond_nl_amli (
    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 507 of file PreCSR.c.

#### 9.76.2.11 fasp\_precond\_schwarz()

```
void fasp_precond_schwarz (
    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 363 of file PreCSR.c.

## 9.76.2.12 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 41 of file PreCSR.c.

## 9.77 PreDataInit.c File Reference

Initialize important data structures.

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

## Functions

- void [fasp\\_precond\\_null](#) ([precond](#) \*pcdata)  
*Initialize precondition data.*
- void [fasp\\_precond\\_data\\_null](#) ([precond\\_data](#) \*pcdata)  
*Initialize precondition 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 memory space.*
- [AMG\\_data\\_bsr](#) \* [fasp\\_amg\\_data\\_bsr\\_create](#) ([SHORT](#) max\_levels)  
*Create and initialize AMG\_data data structure for AMG/SAMG (BSR format)*
- void [fasp\\_amg\\_data\\_bsr\\_free](#) ([AMG\\_data\\_bsr](#) \*mgl)  
*Free AMG\_data\_bsr data memory 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 structure.*
- void [fasp\\_ilu\\_data\\_null](#) ([ILU\\_data](#) \*ILUdata)  
*Initialize ILU data.*
- void [fasp\\_schwarz\\_data\\_free](#) ([Schwarz\\_data](#) \*Schwarz)  
*Free Schwarz\_data data memory space.*

### 9.77.1 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](#)

#### Warning

Every structures should be initialized before usage.

### 9.77.2 Function Documentation

#### 9.77.2.1 fasp\_amg\_data\_bsr\_create()

```
AMG\_data\_bsr * fasp_amg_data_bsr_create (
    SHORT max_levels )
```

Create and initialize [AMG\\_data](#) data structure 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 181 of file PreDataInit.c.

## 9.77.2.2 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

## Date

2013/02/13

Definition at line 211 of file PreDataInit.c.

## 9.77.2.3 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 75 of file PreDataInit.c.

**9.77.2.4 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 internal memory for Intel MKL PARDISO.

Definition at line 107 of file PreDataInit.c.



## 9.77.2.5 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

Definition at line 257 of file PreDataInit.c.

## 9.77.2.6 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

Definition at line 287 of file PreDataInit.c.

### 9.77.2.7 fasp\_ilu\_data\_null()

```
void fasp_ilu_data_null (
    ILU_data * ILUdata )
```

Initialize ILU data.

#### Parameters

<i>ILUdata</i>	Pointer to <a href="#">ILU_data</a>
----------------	-------------------------------------

#### Author

Chensong Zhang

#### Date

2010/03/23

Definition at line 312 of file PreDataInit.c.

### 9.77.2.8 fasp\_precond\_data\_null()

```
void fasp_precond_data_null (
    precondition_data * pcddata )
```

Initialize [precond\\_data](#).

#### Parameters

<i>pcdata</i>	Preconditioning data structure
---------------	--------------------------------

#### Author

Chensong Zhang

#### Date

2010/03/23

Definition at line 44 of file PreDataInit.c.

### 9.77.2.9 fasp\_precond\_null()

```
void fasp_precond_null (
    precondition * pcddata )
```

Initialize precondition data.

#### Parameters

<i>pcddata</i>	Pointer to precondition
----------------	-------------------------

#### Author

Chensong Zhang

#### Date

2010/03/23

Definition at line 28 of file PreDataInit.c.

### 9.77.2.10 fasp\_schwarz\_data\_free()

```
void fasp_schwarz_data_free (
    Schwarz_data * Schwarz )
```

Free [Schwarz\\_data](#) data memory space.

#### Parameters

* <i>Schwarz</i>	pointer to the <a href="#">AMG_data</a> data
------------------	--

#### Author

Xiaozhe Hu

#### Date

2010/04/06

Definition at line 327 of file PreDataInit.c.

## 9.78 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 "PreMGSmoothing.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.*

### 9.78.1 Detailed Description

Abstract multigrid cycle – non-recursive version.

#### Note

This file contains Level-4 (Pre) functions. It requires [AuxArray.c](#), [AuxMessage.c](#), [AuxVector.c](#), [BlaSchwarzSetup.c](#), [BlaArray.c](#), [BlaSpmvBSR.c](#), [BlaSpmvCSR.c](#), [ltrSmootherBSR.c](#), [ltrSmootherCSR.c](#), [ltrSmootherCSRpoly.c](#), [KryPcg.c](#), [KryPvgmres.c](#), [KrySPcg.c](#), and [KrySPvgmres.c](#)

### 9.78.2 Function Documentation

#### 9.78.2.1 [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 12/13/2011 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 52 of file PreMGCycle.c.

**9.78.2.2 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 276 of file PreMGCycle.c.

**9.79 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.*

### 9.79.1 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](#)

### 9.79.2 Function Documentation

#### 9.79.2.1 [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 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 44 of file [PreMGCycleFull.c](#).

## 9.80 PreMGRecur.c File Reference

Abstract multigrid cycle – recursive version.

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

### 9.80.1 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. Will be removed! –Chensong

### 9.80.2 Function Documentation

#### 9.80.2.1 [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 01/10/2012 Modified by Chensong Zhang on 02/27/2013: update direct solvers.

Definition at line 43 of file PreMGRecur.c.

## 9.81 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 "PreMGSmooother.inl"
#include "PreMGRecurAMLI.inl"
```

### Functions

- void [fasp\\_solver\\_amli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) level)  
*Solve  $Ax=b$  with recursive AMLI-cycle.*
- void [fasp\\_solver\\_nl\\_amli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) level, [INT](#) num\_levels)  
*Solve  $Ax=b$  with recursive nonlinear AMLI-cycle.*
- void [fasp\\_solver\\_nl\\_amli\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param, [INT](#) level, [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.*



### 9.81.1 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 AMLI and non-linear AMLI cycles.

### 9.81.2 Function Documentation

#### 9.81.2.1 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 714 of file PreMGRecurAMLI.c.

### 9.81.2.2 fasp\_solver\_amli()

```
void fasp_solver_amli (
    AMG_data * mgl,
    AMG_param * param,
    INT level )
```

Solve  $Ax=b$  with recursive 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>level</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 53 of file PreMGRecurAMLI.c.

### 9.81.2.3 fasp\_solver\_nl\_amli()

```
void fasp_solver_nl_amli (
    AMG_data * mgl,
    AMG_param * param,
    INT level,
    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>level</i>	Current level
<i>num_levels</i>	Total number of levels

## Author

Xiaozhe Hu

## Date

04/06/2010

## Note

Refer to Xiaozhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AML↔I-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 277 of file PreMGRecurAMLI.c.

## 9.81.2.4 fasp\_solver\_nl\_amli\_bsr()

```
void fasp_solver_nl_amli_bsr (
    AMG\_data\_bsr * mgl,
    AMG\_param * param,
    INT level,
    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>level</i>	Current level
<i>num_levels</i>	Total number of levels

**Author**

Xiaozhe Hu

**Date**

04/06/2010

**Note**

Nonlinear AMLI-cycle. Refer to Xiaozhe 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 516 of file PreMGRecurAMLI.c.

## 9.82 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\\_nl\\_amli](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Nonlinear AMLI – SOLVE phase.*
- [void fasp\\_famg\\_solve](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*FMG – SOLVE phase.*

### 9.82.1 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](#)

## 9.82.2 Function Documentation

### 9.82.2.1 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 44 of file PreMGSolve.c.

### 9.82.2.2 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

**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 04/21/2013: Fix an output typo

Definition at line 133 of file PreMGsSolve.c.

**9.82.2.3 fasp\_amg\_solve\_nl\_amli()**

```
INT fasp_amg_solve_nl_amli (  
    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 217 of file PreMGSolve.c.

**9.82.2.4 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 289 of file PreMGSolve.c.

**9.83 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  $z=inv(D)*r$ .*
- 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)*

### 9.83.1 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](#)

### 9.83.2 Function Documentation

#### 9.83.2.1 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 1710 of file PreSTR.c.

**9.83.2.2 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 31 of file PreSTR.c.

**9.83.2.3 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 58 of file PreSTR.c.

**9.83.2.4 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 982 of file PreSTR.c.

## 9.83.2.5 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 819 of file PreSTR.c.

## 9.83.2.6 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 340 of file PreSTR.c.

**9.83.2.7 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 1429 of file PreSTR.c.

**9.83.2.8 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 1163 of file PreSTR.c.

## 9.84 SolAMG.c File Reference

AMG method as an iterative solver.

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

**Functions**

- void [fasp\\_solver\\_amg](#) (const [dCSRmat](#) \*A, const [dvector](#) \*b, [dvector](#) \*x, [AMG\\_param](#) \*param)  
*Solve  $Ax = b$  by algebraic multigrid methods.*

### 9.84.1 Detailed Description

AMG method as an iterative solver.

**Note**

This file contains Level-5 (Sol) functions. It requires [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSparseCSR.c](#), [KrySPgmres.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreDataInit.c](#), and [PreMGSolve.c](#)

### 9.84.2 Function Documentation

#### 9.84.2.1 [fasp\\_solver\\_amg\(\)](#)

```
void 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 dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>param</i>	Pointer to <a href="#">AMG_param</a> : AMG parameters

## 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 01/10/2012 Modified by Chensong Zhang on 07/26/2014: Add error handling for AMG setup

Definition at line 42 of file SolAMG.c.

## 9.85 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, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov\\_block\\_3](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, [dCSRmat](#) \*A\_diag)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dblc\\_krylov\\_block\\_4](#) ([dBLCmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_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, [itsolver\\_param](#) \*itparam, [INT](#) NumLayers, [dBLCmat](#) \*Ai, [dCSRmat](#) \*local\_A, [ivector](#) \*local\_index)  
*Solve  $Ax = b$  by standard Krylov methods.*

### 9.85.1 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](#), [KryPvbcgs.c](#), [KryPvfgmres.c](#), [KryPvgmres.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreBLC.c](#), and [PreDataInit.c](#)

### 9.85.2 Function Documentation

#### 9.85.2.1 fasp\_solver\_dblc\_itsolver()

```
INT fasp_solver_dblc_itsolver (
    dBLCmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    itsolver_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 49 of file SolBLC.c.

### 9.85.2.2 fasp\_solver\_dblc\_krylov()

```

INT fasp_solver_dblc_krylov (
    dBLCmat * A,
    dvector * b,
    dvector * x,
    itsolver_param * itparam )

```

Solve  $Ax = b$  by standard Krylov methods.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <b>dBLCmat</b> 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 141 of file SolBLC.c.

### 9.85.2.3 fasp\_solver\_dblc\_krylov\_block\_3()

```

INT fasp_solver_dblc_krylov_block_3 (
    dBLCmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 3by3 block [dCSRmat](#) problems!! – Xiaozhe Hu

Definition at line 195 of file SolBLC.c.

## 9.85.2.4 fasp\_solver\_dblc\_krylov\_block\_4()

```

INT fasp_solver_dblc_krylov_block_4 (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    itsolver_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 392 of file SolBLC.c.

**9.85.2.5 fasp\_solver\_dblc\_krylov\_sweeping()**

```

INT fasp_solver_dblc_krylov_sweeping (
    dBLCMat * A,
    dvector * b,
    dvector * x,
    itsolver_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
<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 518 of file SolBLC.c.

## 9.86 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, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for BSR matrices.*
- [INT fasp\\_solver\\_dbsr\\_krylov](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods for BSR matrices.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_diag](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dbsr\\_krylov\\_ilu](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_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, [itsolver\\_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, [itsolver\\_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, [itsolver\\_param](#) \*itparam, [AMG\\_param](#) \*amgparam, const [INT](#) nk\_dim, [dvector](#) \*nk)  
*Solve  $Ax=b$  by AMG preconditioned Krylov methods with extra kernal space.*

### 9.86.1 Detailed Description

Iterative solvers for [dBSRmat](#) matrices.

#### Note

This file contains Level-5 (Sol) functions. It requires [AuxMemory.c](#), [AuxMessage.c](#), [AuxTiming.c](#), [AuxVector.c](#), [BlaSmallMatInv.c](#), [BlalLUSetupBSR.c](#), [BlaSparseBSR.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgmres.c](#), [KryPvbcgs.c](#), [KryPvfgmres.c](#), [KryPvgmres.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreBSR.c](#), and [PreDataInit.c](#)

### 9.86.2 Function Documentation

#### 9.86.2.1 fasp\_solver\_dbsr\_itsolver()

```
INT fasp_solver_dbsr_itsolver (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    itsolver_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 49 of file SolBSR.c.

## 9.86.2.2 fasp\_solver\_dbsr\_krylov()

```

INT fasp_solver_dbsr_krylov (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 142 of file SolBSR.c.

## 9.86.2.3 fasp\_solver\_dbsr\_krylov\_amg()

```

INT fasp_solver_dbsr_krylov_amg (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 364 of file SolBSR.c.

**9.86.2.4 fasp\_solver\_dbsr\_krylov\_amg\_nk()**

```
INT fasp_solver_dbsr_krylov_amg_nk (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 506 of file SolBSR.c.

**9.86.2.5 fasp\_solver\_dbsr\_krylov\_diag()**

```
INT fasp_solver_dbsr_krylov_diag (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 193 of file SolBSR.c.

### 9.86.2.6 fasp\_solver\_dbsr\_krylov\_ilu()

```

INT fasp_solver_dbsr_krylov_ilu (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 297 of file SolBSR.c.

### 9.86.2.7 fasp\_solver\_dbsr\_krylov\_nk\_amg()

```

INT fasp_solver_dbsr_krylov_nk_amg (
    dBSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 665 of file SolBSR.c.

## 9.87 SolCSR.c File Reference

Iterative solvers for [dCSRmat](#) matrices.

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

## Functions

- [INT fasp\\_solver\\_dcsr\\_itsolver](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.*
- [INT fasp\\_solver\\_dcsr\\_krylov](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods for CSR matrices.*
- [INT fasp\\_solver\\_dcsr\\_krylov\\_diag](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*

- `INT fasp_solver_dcsr_krylov_schwarz (dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, Schwarz_param *schparam)`  
Solve  $Ax=b$  by overlapping Schwarz Krylov methods.
- `INT fasp_solver_dcsr_krylov_amg (dCSRmat *A, dvector *b, dvector *x, itsolver_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, itsolver_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, itsolver_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, itsolver_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.

### 9.87.1 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`, `BlaSparseCSR.c`, `KryPbcgs.c`, `KryPcg.c`, `KryPgcg.c`, `KryPgcr.c`, `KryPgmres.c`, `KryPminres.c`, `KryPvbcgs.c`, `KryPvfgmres.c`, `KryPvgmres.c`, `PreAMGSetupRS.c`, `PreAMGSetupSA.c`, `PreAMGSetupUA.c`, `PreCSR.c`, and `PreDataInit.c`

### 9.87.2 Function Documentation

#### 9.87.2.1 fasp\_solver\_dcsr\_itsolver()

```
INT fasp_solver_dcsr_itsolver (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    itsolver_param * itparam )
```

Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <code>dCSRmat</code> 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

**Note**

This is an abstract interface for iterative methods. Modified by Chunsheng Feng on 03/04/2016: add VBiCGstab solver

Definition at line 52 of file SolCSR.c.

**9.87.2.2 fasp\_solver\_dcsr\_krylov()**

```
INT fasp_solver_dcsr_krylov (  
    dCSRmat * A,  
    dvector * b,  
    dvector * x,  
    itsolver_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 164 of file SolCSR.c.

## 9.87.2.3 fasp\_solver\_dcsr\_krylov\_amg()

```
INT fasp_solver_dcsr_krylov_amg (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 359 of file SolCSR.c.

## 9.87.2.4 fasp\_solver\_dcsr\_krylov\_amg\_nk()

```

INT fasp_solver_dcsr_krylov_amg_nk (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 632 of file SolCSR.c.

## 9.87.2.5 fasp\_solver\_dcsr\_krylov\_diag()

```

INT fasp_solver_dcsr_krylov_diag (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 214 of file SolCSR.c.

**9.87.2.6 fasp\_solver\_dcsr\_krylov\_ilu()**

```

INT fasp_solver_dcsr_krylov_ilu (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 464 of file SolCSR.c.

**9.87.2.7 fasp\_solver\_dcsr\_krylov\_ilu\_M()**

```
INT fasp_solver_dcsr_krylov_ilu_M (  
    dCSRmat * A,  
    dvector * b,  
    dvector * x,  
    itsolver_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 548 of file SolCSR.c.

### 9.87.2.8 fasp\_solver\_dcsr\_krylov\_schwarz()

```
INT fasp_solver_dcsr_krylov_schwarz (
    dCSRmat * A,
    dvector * b,
    dvector * x,
    itsolver_param * itparam,
    Schwarz_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 278 of file SolCSR.c.

## 9.88 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.*

### 9.88.1 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](#), [BlaSparseCSR.c](#), [PreAMGSetupRS.c](#), [PreAMGSetupSA.c](#), [PreAMGSetupUA.c](#), [PreDataInit.c](#), and [PreMGsSolve.c](#)

### 9.88.2 Function Documentation

#### 9.88.2.1 fasp\_solver\_famg()

```
void 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 01/10/2012 Modified by Chensong Zhang on 05/05/2013: Remove error handling for AMG setup

Definition at line 36 of file SolFAMG.c.

## 9.89 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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*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 prtlvl](#))  
*Solve  $Ax=b$  of Poisson 3D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)*

### 9.89.1 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](#)

## 9.89.2 Function Documentation

### 9.89.2.1 fasp\_poisson\_fgmg1d()

```
void fasp_poisson_fgmg1d (
    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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 445 of file SolGMGPoisson.c.

### 9.89.2.2 fasp\_poisson\_fgmg2d()

```
void fasp_poisson_fgmg2d (
    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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 540 of file SolGMGPoisson.c.

## 9.89.2.3 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 649 of file SolGMGPoisson.c.

**9.89.2.4 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 45 of file SolGMGPoisson.c.

### 9.89.2.5 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>prtvl</i>	Print level for output

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Ziteng Wang, Chensong Zhang

#### Date

06/07/2013

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 171 of file SolGMGPoisson.c.

### 9.89.2.6 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 309 of file SolGMGPoisson.c.

## 9.89.2.7 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 760 of file SolGMGPoisson.c.

**9.89.2.8 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 856 of file SolGMGPoisson.c.

**9.89.2.9 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

Modified by Chensong Zhang on 01/14/2017: Clean up

Definition at line 967 of file SolGMGPoisson.c.

## 9.90 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, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.*
- [INT fasp\\_solver\\_krylov](#) ([mxv\\_matfree](#) \*mf, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_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.*

### 9.90.1 Detailed Description

Iterative solvers using MatFree spmv operations.

#### Note

This file contains Level-5 (Sol) functions. It requires [AuxMessage.c](#), [AuxTiming.c](#), [BlaSpmvBLC.c](#), [BlaSpmvB↵SR.c](#), [BlaSpmvCSR.c](#), [BlaSpmvCSRL.c](#), [BlaSpmvSTR.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgcg.c](#), [KryPgmres.c](#), [Kry↵Pminres.c](#), [KryPvfgmres.c](#), and [KryPvgmres.c](#)

### 9.90.2 Function Documentation

#### 9.90.2.1 fasp\_solver\_itsolver()

```
INT fasp_solver_itsolver (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    precondition * pc,
    itsolver_param * itparam )
```

Solve  $Ax=b$  by preconditioned Krylov methods for CSR matrices.

## 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>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

## Note

This is an abstract interface for iterative methods.

Modified by Feiteng Huang on 09/19/2012: matrix free

Definition at line 53 of file SolMatFree.c.

## 9.90.2.2 fasp\_solver\_krylov()

```

INT fasp_solver_krylov (
    mxv_matfree * mf,
    dvector * b,
    dvector * x,
    itsolver_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 158 of file SolMatFree.c.

**9.90.2.3 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 Modified by Chensong Zhang on 01/20/2017

Definition at line 206 of file SolMatFree.c.

## 9.91 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, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov\\_diag](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*
- [INT fasp\\_solver\\_dstr\\_krylov\\_ilu](#) ([dSTRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_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, [itsolver\\_param](#) \*itparam, [ivector](#) \*neigh, [ivector](#) \*order)  
*Solve  $Ax=b$  by diagonal preconditioned Krylov methods.*

### 9.91.1 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](#), [BlaILUSetupSTR.c](#), [BlaSparseSTR.c](#), [ItrSmootherSTR.c](#), [KryPbcgs.c](#), [KryPcg.c](#), [KryPgmres.c](#), [KryPvbcgs.c](#), [KryPvgmres.c](#), and [PreSTR.c](#)

### 9.91.2 Function Documentation

#### 9.91.2.1 fasp\_solver\_dstr\_itsolver()

```
INT fasp_solver_dstr_itsolver (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    precondition * pc,
    itsolver_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 46 of file SolSTR.c.

**9.91.2.2 fasp\_solver\_dstr\_krylov()**

```

INT fasp_solver_dstr_krylov (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 134 of file SolSTR.c.

**9.91.2.3 fasp\_solver\_dstr\_krylov\_blockgs()**

```
INT fasp_solver_dstr_krylov_blockgs (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 341 of file SolSTR.c.

#### 9.91.2.4 fasp\_solver\_dstr\_krylov\_diag()

```

INT fasp_solver_dstr_krylov_diag (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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 182 of file SolSTR.c.

#### 9.91.2.5 fasp\_solver\_dstr\_krylov\_ilu()

```

INT fasp_solver_dstr_krylov_ilu (
    dSTRmat * A,
    dvector * b,
    dvector * x,
    itsolver_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
<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 248 of file SolSTR.c.

## 9.92 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\\_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\\_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.*
- INT [fasp\\_wrapper\\_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)  
*Solve  $Ax=b$  by Krylov method preconditioned by AMG (dcsr -> dbsr)*
- INT [fasp\\_wrapper\\_dcoo\\_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)  
*Solve  $Ax=b$  by Krylov method preconditioned by AMG (dcoo -> dbsr)*

### 9.92.1 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](#)

## 9.92.2 Function Documentation

### 9.92.2.1 fasp\_fwrapper\_amg\_()

```
void fasp_fwrapper_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 39 of file SolWrapper.c.

## 9.92.2.2 fasp\_fwrapper\_krylov\_amg\_()

```

void fasp_fwrapper_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 89 of file SolWrapper.c.

## 9.92.2.3 fasp\_wrapper\_dbsr\_krylov\_amg()

```

INT fasp_wrapper_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 )

```

Solve  $Ax=b$  by Krylov method preconditioned by AMG (dcsr - > dbcsr)

#### Parameters

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>nb</i>	Size of each small block
<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

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xiaozhe Hu

#### Date

03/05/2013

Definition at line 156 of file SolWrapper.c.

#### 9.92.2.4 fasp\_wrapper\_dcoo\_dbcsr\_krylov\_amg()

```

INT fasp_wrapper_dcoo_dbcsr_krylov_amg (
    INT n,
    INT nnz,
    INT nb,
    INT * ia,
    INT * ja,

```

```

REAL * a,
REAL * b,
REAL * u,
REAL tol,
INT maxit,
INT ptrlvl )

```

Solve  $Ax=b$  by Krylov method preconditioned by AMG (dcoo - > dbcsr)

#### Parameters

<i>n</i>	Number of cols of A
<i>nnz</i>	Number of nonzeros of A
<i>nb</i>	Size of each small block
<i>ia</i>	IA of A in COO format
<i>ja</i>	JA of A in COO format
<i>a</i>	VAL of A in COO 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

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Xiaozhe Hu

#### Date

03/06/2013

Definition at line 242 of file SolWrapper.c.

## 9.93 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.*

### 9.93.1 Detailed Description

Interface to MUMPS direct solvers.

Reference for MUMPS: <http://mumps.enseeiht.fr/>

### 9.93.2 Macro Definition Documentation

#### 9.93.2.1 ICNTL

```
#define ICNTL(  
    I ) icntl[(I)-1]
```

macro s.t. indices match documentation

Definition at line 18 of file XtrMumps.c.

### 9.93.3 Function Documentation

#### 9.93.3.1 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 40 of file XtrMumps.c.

## 9.93.3.2 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.

Definition at line 170 of file XtrMumps.c.

## 9.94 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. Each row of A should be in ascending order w.r.t. column indices.*

#### 9.94.1 Detailed Description

Interface to Intel MKL PARDISO direct solvers.

Reference for Intel MKL PARDISO: <https://software.intel.com/en-us/node/470282>

#### 9.94.2 Function Documentation

##### 9.94.2.1 fasp\_solver\_pardiso()

```
int fasp_solver_pardiso (
    dCSRmat * ptrA,
    dvector * b,
    dvector * u,
    const SHORT prtlvl )
```

Solve  $Ax=b$  by PARDISO directly. Each row of A should be in ascending order w.r.t. column indices.

##### 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**

Hongxuan Zhang

**Date**

11/28/2015

Definition at line 39 of file XtrPardiso.c.

## 9.95 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.*

### 9.95.1 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)!

### 9.95.2 Function Documentation

#### 9.95.2.1 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 <a href="#">dCSRmat</a> 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 60 of file XtrSamg.c.

**9.95.2.2 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 31 of file XtrSamg.c.

**9.96 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.*

### 9.96.1 Detailed Description

Interface to SuperLU direct solvers.

Reference for SuperLU: <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>

### 9.96.2 Function Documentation

#### 9.96.2.1 `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 <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

Xiaozhe Hu

## Date

11/05/09

Modified by Chensong Zhang on 11/01/2012 for new FASP function names. Modified by Chensong Zhang on 02/27/2013 for new FASP function names.

Definition at line 41 of file XtrSuperlu.c.

## 9.97 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.*

#### 9.97.1 Detailed Description

Interface to UMFPACK direct solvers.

Reference for SuiteSparse: <http://faculty.cse.tamu.edu/davis/suitesparse.html>

#### 9.97.2 Function Documentation

##### 9.97.2.1 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 38 of file XtrUmfpack.c.



# Index

\_\_FASPBLOCK\_HEADER\_\_  
    fasp\_block.h, [325](#)  
\_\_FASPGRID\_HEADER\_\_  
    fasp\_grid.h, [360](#)  
\_\_FASP\_HEADER\_\_  
    fasp.h, [315](#)

## A

    precond\_sweeping\_data, [69](#)

## A\_diag

    precond\_block\_data, [61](#)

## ABS

    fasp.h, [315](#)

## AMG\_ILU\_levels

    input\_param, [44](#)

## AMG\_Schwarz\_levels

    input\_param, [46](#)

## AMG\_aggregation\_type

    input\_param, [42](#)

## AMG\_aggressive\_level

    input\_param, [42](#)

## AMG\_aggressive\_path

    input\_param, [42](#)

## AMG\_amli\_degree

    input\_param, [42](#)

## AMG\_coarse\_dof

    input\_param, [43](#)

## AMG\_coarse\_scaling

    input\_param, [43](#)

## AMG\_coarse\_solver

    input\_param, [43](#)

## AMG\_coarsening\_type

    input\_param, [43](#)

## AMG\_cycle\_type

    input\_param, [43](#)

## AMG\_data, [19](#)

## AMG\_data\_bsr, [20](#)

## AMG\_interpolation\_type

    input\_param, [44](#)

## AMG\_levels

    input\_param, [44](#)

## AMG\_max\_aggregation

    input\_param, [44](#)

## AMG\_max\_row\_sum

    input\_param, [44](#)

## AMG\_maxit

    input\_param, [45](#)

## AMG\_nl\_amli\_krylov\_type

    input\_param, [45](#)

## AMG\_pair\_number

    input\_param, [45](#)

## AMG\_param, [22](#)

## AMG\_polynomial\_degree

    input\_param, [45](#)

## AMG\_postsmooth\_iter

    input\_param, [45](#)

## AMG\_presmooth\_iter

    input\_param, [46](#)

## AMG\_quality\_bound

    input\_param, [46](#)

## AMG\_relaxation

    input\_param, [46](#)

## AMG\_smooth\_filter

    input\_param, [46](#)

## AMG\_smooth\_order

    input\_param, [47](#)

## AMG\_smoother

    input\_param, [47](#)

## AMG\_strong\_coupled

    input\_param, [47](#)

## AMG\_strong\_threshold

    input\_param, [47](#)

## AMG\_tentative\_smooth

    input\_param, [47](#)

## AMG\_tol

    input\_param, [48](#)

## AMG\_truncation\_threshold

    input\_param, [48](#)

## AMG\_type

    input\_param, [48](#)

## AMLI\_CYCLE

    fasp\_const.h, [330](#)

## ASCEND

    fasp\_const.h, [330](#)

## Ablc

    precond\_block\_data, [61](#)

## Ai

    precond\_sweeping\_data, [69](#)

## amgparam

    precond\_block\_data, [61](#)

## AuxArray.c, [75](#)

- fasp\_array\_cp, 76
  - fasp\_array\_cp\_nc3, 76
  - fasp\_array\_cp\_nc5, 77
  - fasp\_array\_cp\_nc7, 77
  - fasp\_array\_null, 78
  - fasp\_array\_set, 79
  - fasp\_iarray\_cp, 79
  - fasp\_iarray\_set, 80
- AuxConvert.c, 81
  - fasp\_aux\_bbyteToldouble, 81
  - fasp\_aux\_change\_endian4, 82
  - fasp\_aux\_change\_endian8, 82
- AuxGivens.c, 83
  - fasp\_aux\_givens, 83
- AuxGraphics.c, 84
  - fasp\_dbsr\_plot, 85
  - fasp\_dbsr\_subplot, 85
  - fasp\_dcsr\_plot, 86
  - fasp\_dcsr\_subplot, 87
  - fasp\_grid2d\_plot, 87
- AuxInput.c, 88
  - fasp\_param\_check, 89
  - fasp\_param\_input, 89
- AuxMemory.c, 90
  - fasp\_mem\_calloc, 91
  - fasp\_mem\_check, 91
  - fasp\_mem\_free, 92
  - fasp\_mem\_iludata\_check, 93
  - fasp\_mem\_realloc, 93
  - fasp\_mem\_usage, 94
  - total\_alloc\_count, 94
  - total\_alloc\_mem, 95
- AuxMessage.c, 95
  - fasp\_chkerr, 96
  - print\_amgcomplexity, 96
  - print\_amgcomplexity\_bsr, 97
  - print\_cputime, 97
  - print\_itinfo, 98
  - print\_message, 99
- AuxParam.c, 99
  - fasp\_param\_amg\_init, 101
  - fasp\_param\_amg\_print, 101
  - fasp\_param\_amg\_set, 102
  - fasp\_param\_amg\_to\_prec, 102
  - fasp\_param\_amg\_to\_prec\_bsr, 103
  - fasp\_param\_ilu\_init, 103
  - fasp\_param\_ilu\_print, 104
  - fasp\_param\_ilu\_set, 104
  - fasp\_param\_init, 105
  - fasp\_param\_input\_init, 105
  - fasp\_param\_prec\_to\_amg, 106
  - fasp\_param\_prec\_to\_amg\_bsr, 106
  - fasp\_param\_schwarz\_init, 107
  - fasp\_param\_schwarz\_print, 107
  - fasp\_param\_schwarz\_set, 108
  - fasp\_param\_set, 108
  - fasp\_param\_solver\_init, 109
  - fasp\_param\_solver\_print, 109
  - fasp\_param\_solver\_set, 110
- AuxSort.c, 111
  - fasp\_BinarySearch, 117
  - fasp\_aux\_dQuickSort, 111
  - fasp\_aux\_dQuickSortIndex, 112
  - fasp\_aux\_iQuickSort, 113
  - fasp\_aux\_iQuickSortIndex, 113
  - fasp\_aux\_merge, 115
  - fasp\_aux\_msort, 116
  - fasp\_aux\_unique, 116
  - fasp\_multicolors\_independent\_set, 118
  - fasp\_topological\_sorting\_ilu, 118
- AuxThreads.c, 119
  - fasp\_get\_start\_end, 120
  - fasp\_set\_GS\_threads, 120
  - THDs\_AMG\_GS, 121
  - THDs\_CPR\_gGS, 121
  - THDs\_CPR\_IGS, 121
- AuxTiming.c, 122
  - fasp\_gettime, 122
- AuxVector.c, 123
  - fasp\_dvec\_alloc, 124
  - fasp\_dvec\_cp, 124
  - fasp\_dvec\_create, 125
  - fasp\_dvec\_free, 125
  - fasp\_dvec\_isnan, 126
  - fasp\_dvec\_maxdiff, 126
  - fasp\_dvec\_null, 127
  - fasp\_dvec\_rand, 128
  - fasp\_dvec\_set, 128
  - fasp\_dvec\_symdiagscale, 129
  - fasp\_ivec\_alloc, 130
  - fasp\_ivec\_create, 130
  - fasp\_ivec\_free, 131
  - fasp\_ivec\_set, 131
- BIGREAL
  - fasp\_const.h, 330
- BlaArray.c, 132
  - fasp\_blas\_array\_ax, 133
  - fasp\_blas\_array\_axpby, 133
  - fasp\_blas\_array\_axpy, 134
  - fasp\_blas\_array\_axpyz, 135
  - fasp\_blas\_array\_dotprod, 136
  - fasp\_blas\_array\_norm1, 136
  - fasp\_blas\_array\_norm2, 137
  - fasp\_blas\_array\_norminf, 138
- BlaEigen.c, 139
  - fasp\_dcsr\_eig, 139
- BlaFormat.c, 140



- fasp\_format\_dblc\_dcsr, 141
- fasp\_format\_dbsr\_dcoo, 141
- fasp\_format\_dbsr\_dcsr, 142
- fasp\_format\_dcoo\_dcsr, 143
- fasp\_format\_dcsr\_dbsr, 143
- fasp\_format\_dcsr\_dcoo, 144
- fasp\_format\_dcsr\_dcsr, 145
- fasp\_format\_dstr\_dbsr, 145
- fasp\_format\_dstr\_dcsr, 146
- BlaILU.c, 147
  - fasp\_iluk, 147
  - fasp\_ilut, 149
  - fasp\_ilutp, 150
  - fasp\_symbfactor, 151
- BlaILUSetupBSR.c, 154
  - fasp\_ilu\_dbsr\_setup, 155
  - fasp\_ilu\_dbsr\_setup\_levsch\_omp, 156
  - fasp\_ilu\_dbsr\_setup\_mc\_omp, 156
  - fasp\_ilu\_dbsr\_setup\_omp, 157
- BlaILUSetupCSR.c, 158
  - fasp\_ilu\_dcsr\_setup, 159
- BlaILUSetupSTR.c, 159
  - fasp\_ilu\_dstr\_setup0, 160
  - fasp\_ilu\_dstr\_setup1, 160
- BlaIO.c, 161
  - dlength, 188
  - fasp\_dbsr\_print, 163
  - fasp\_dbsr\_read, 164
  - fasp\_dbsr\_write, 165
  - fasp\_dbsr\_write\_coo, 165
  - fasp\_dcoo1\_read, 166
  - fasp\_dcoo\_print, 167
  - fasp\_dcoo\_read, 167
  - fasp\_dcoo\_shift\_read, 168
  - fasp\_dcoo\_write, 169
  - fasp\_dcsr\_print, 169
  - fasp\_dcsr\_read, 170
  - fasp\_dcsr\_write\_coo, 170
  - fasp\_dcsrvec1\_read, 171
  - fasp\_dcsrvec1\_write, 172
  - fasp\_dcsrvec2\_read, 173
  - fasp\_dcsrvec2\_write, 174
  - fasp\_dmtx\_read, 174
  - fasp\_dmtxsym\_read, 175
  - fasp\_dstr\_print, 176
  - fasp\_dstr\_read, 176
  - fasp\_dstr\_write, 177
  - fasp\_dvec\_print, 178
  - fasp\_dvec\_read, 178
  - fasp\_dvec\_write, 179
  - fasp\_dvecind\_read, 179
  - fasp\_dvecind\_write, 180
  - fasp\_hb\_read, 181
  - fasp\_ivec\_print, 181
  - fasp\_ivec\_read, 182
  - fasp\_ivec\_write, 183
  - fasp\_ivecind\_read, 183
  - fasp\_matrix\_read, 184
  - fasp\_matrix\_read\_bin, 185
  - fasp\_matrix\_write, 185
  - fasp\_vector\_read, 186
  - fasp\_vector\_write, 187
  - ilength, 188
- BlaOrderingCSR.c, 189
  - fasp\_dcsr\_CMK\_order, 189
  - fasp\_dcsr\_RCMK\_order, 190
- BlaSchwarzSetup.c, 190
  - fasp\_dcsr\_schwarz\_backward\_smoother, 191
  - fasp\_dcsr\_schwarz\_forward\_smoother, 192
  - fasp\_schwarz\_setup, 192
- BlaSmallMat.c, 193
  - fasp\_blas\_array\_axpy\_nc2, 195
  - fasp\_blas\_array\_axpy\_nc3, 195
  - fasp\_blas\_array\_axpy\_nc5, 196
  - fasp\_blas\_array\_axpy\_nc7, 197
  - fasp\_blas\_array\_axpyz\_nc2, 197
  - fasp\_blas\_array\_axpyz\_nc3, 198
  - fasp\_blas\_array\_axpyz\_nc5, 199
  - fasp\_blas\_array\_axpyz\_nc7, 199
  - fasp\_blas\_smat\_aAxpby, 200
  - fasp\_blas\_smat\_add, 201
  - fasp\_blas\_smat\_axm, 202
  - fasp\_blas\_smat\_mul, 202
  - fasp\_blas\_smat\_mul\_nc2, 203
  - fasp\_blas\_smat\_mul\_nc3, 203
  - fasp\_blas\_smat\_mul\_nc5, 204
  - fasp\_blas\_smat\_mul\_nc7, 205
  - fasp\_blas\_smat\_m xv, 205
  - fasp\_blas\_smat\_m xv\_nc2, 206
  - fasp\_blas\_smat\_m xv\_nc3, 206
  - fasp\_blas\_smat\_m xv\_nc5, 207
  - fasp\_blas\_smat\_m xv\_nc7, 208
  - fasp\_blas\_smat\_ymAx, 208
  - fasp\_blas\_smat\_ymAx\_nc2, 209
  - fasp\_blas\_smat\_ymAx\_nc3, 210
  - fasp\_blas\_smat\_ymAx\_nc5, 210
  - fasp\_blas\_smat\_ymAx\_nc7, 211
  - fasp\_blas\_smat\_ypAx, 212
  - fasp\_blas\_smat\_ypAx\_nc2, 212
  - fasp\_blas\_smat\_ypAx\_nc3, 213
  - fasp\_blas\_smat\_ypAx\_nc5, 213
  - fasp\_blas\_smat\_ypAx\_nc7, 214
- BlaSmallMatInv.c, 215
  - fasp\_smat\_Linfinity, 223
  - fasp\_smat\_identity, 216
  - fasp\_smat\_identity\_nc2, 217
  - fasp\_smat\_identity\_nc3, 217
  - fasp\_smat\_identity\_nc5, 218

- fasp\_smat\_identity\_nc7, 218
  - fasp\_smat\_inv, 219
  - fasp\_smat\_inv\_nc, 219
  - fasp\_smat\_inv\_nc2, 220
  - fasp\_smat\_inv\_nc3, 220
  - fasp\_smat\_inv\_nc4, 221
  - fasp\_smat\_inv\_nc5, 221
  - fasp\_smat\_inv\_nc7, 222
  - fasp\_smat\_invp\_nc, 222
  - SWAP, 216
- BlaSmallMatLU.c, 224
  - fasp\_smat\_lu\_decomp, 224
  - fasp\_smat\_lu\_solve, 225
- BlaSparseBLC.c, 226
  - fasp\_dblc\_free, 227
  - fasp\_dbsr\_getblk, 227
- BlaSparseBSR.c, 228
  - fasp\_dbsr\_alloc, 229
  - fasp\_dbsr\_cp, 230
  - fasp\_dbsr\_create, 230
  - fasp\_dbsr\_diagLU2, 234
  - fasp\_dbsr\_diagLU, 234
  - fasp\_dbsr\_diaginv, 231
  - fasp\_dbsr\_diaginv2, 232
  - fasp\_dbsr\_diaginv3, 232
  - fasp\_dbsr\_diaginv4, 233
  - fasp\_dbsr\_diagpref, 235
  - fasp\_dbsr\_free, 236
  - fasp\_dbsr\_getdiag, 236
  - fasp\_dbsr\_getdiaginv, 237
  - fasp\_dbsr\_null, 238
  - fasp\_dbsr\_perm, 238
  - fasp\_dbsr\_trans, 239
- BlaSparseCOO.c, 244
  - fasp\_dcoo\_alloc, 245
  - fasp\_dcoo\_create, 245
  - fasp\_dcoo\_free, 246
  - fasp\_dcoo\_shift, 246
- BlaSparseCSR.c, 247
  - fasp\_dcsr\_alloc, 249
  - fasp\_dcsr\_bandwidth, 249
  - fasp\_dcsr\_compress, 250
  - fasp\_dcsr\_compress\_inplace, 250
  - fasp\_dcsr\_cp, 251
  - fasp\_dcsr\_create, 252
  - fasp\_dcsr\_diagpref, 252
  - fasp\_dcsr\_free, 253
  - fasp\_dcsr\_getblk, 254
  - fasp\_dcsr\_getcol, 254
  - fasp\_dcsr\_getdiag, 255
  - fasp\_dcsr\_multicoloring, 256
  - fasp\_dcsr\_null, 256
  - fasp\_dcsr\_perm, 257
  - fasp\_dcsr\_permz, 257
  - fasp\_dcsr\_regdiag, 258
  - fasp\_dcsr\_shift, 259
  - fasp\_dcsr\_sort, 259
  - fasp\_dcsr\_sortz, 260
  - fasp\_dcsr\_symdiagscale, 260
  - fasp\_dcsr\_sympart, 261
  - fasp\_dcsr\_trans, 262
  - fasp\_dcsr\_transz, 262
  - fasp\_icsr\_cp, 263
  - fasp\_icsr\_create, 263
  - fasp\_icsr\_free, 264
  - fasp\_icsr\_null, 265
  - fasp\_icsr\_trans, 265
- BlaSparseCSRL.c, 266
  - fasp\_dcsrl\_create, 266
  - fasp\_dcsrl\_free, 267
- BlaSparseCheck.c, 240
  - fasp\_check\_dCSRmat, 240
  - fasp\_check\_diagdom, 241
  - fasp\_check\_diagpos, 241
  - fasp\_check\_diagzero, 242
  - fasp\_check\_iCSRmat, 243
  - fasp\_check\_symm, 243
- BlaSparseSTR.c, 267
  - fasp\_dstr\_alloc, 268
  - fasp\_dstr\_cp, 269
  - fasp\_dstr\_create, 269
  - fasp\_dstr\_free, 270
  - fasp\_dstr\_null, 270
- BlaSparseUtil.c, 271
  - fasp\_sparse\_MIS, 277
  - fasp\_sparse\_aat\_, 272
  - fasp\_sparse\_abyb\_, 273
  - fasp\_sparse\_abybms\_, 274
  - fasp\_sparse\_aplbms\_, 275
  - fasp\_sparse\_aplusb\_, 275
  - fasp\_sparse\_iit\_, 276
  - fasp\_sparse\_rapcmp\_, 277
  - fasp\_sparse\_rapms\_, 278
  - fasp\_sparse\_wta\_, 279
  - fasp\_sparse\_wtams\_, 280
  - fasp\_sparse\_ytx\_, 281
  - fasp\_sparse\_ytxbig\_, 281
- BlaSpmvBLC.c, 282
  - fasp\_blas\_dblc\_aApy, 282
  - fasp\_blas\_dblc\_mxv, 283
- BlaSpmvBSR.c, 284
  - fasp\_blas\_dbsr\_aApyby, 284
  - fasp\_blas\_dbsr\_aApy, 285
  - fasp\_blas\_dbsr\_aApy\_agg, 286
  - fasp\_blas\_dbsr\_axm, 287
  - fasp\_blas\_dbsr\_mxm, 287
  - fasp\_blas\_dbsr\_mxv, 288
  - fasp\_blas\_dbsr\_mxv\_agg, 289

- [fasp\\_blas\\_dbsr\\_rap](#), [289](#)
  - [fasp\\_blas\\_dbsr\\_rap1](#), [290](#)
  - [fasp\\_blas\\_dbsr\\_rap\\_agg](#), [291](#)
- [BlaSpmvCSR.c](#), [291](#)
  - [fasp\\_blas\\_dcsr\\_aAxy](#), [293](#)
  - [fasp\\_blas\\_dcsr\\_aAxy\\_agg](#), [293](#)
  - [fasp\\_blas\\_dcsr\\_add](#), [294](#)
  - [fasp\\_blas\\_dcsr\\_axm](#), [295](#)
  - [fasp\\_blas\\_dcsr\\_mxm](#), [295](#)
  - [fasp\\_blas\\_dcsr\\_mxv](#), [296](#)
  - [fasp\\_blas\\_dcsr\\_mxv\\_agg](#), [297](#)
  - [fasp\\_blas\\_dcsr\\_ptap](#), [297](#)
  - [fasp\\_blas\\_dcsr\\_rap](#), [298](#)
  - [fasp\\_blas\\_dcsr\\_rap2](#), [299](#)
  - [fasp\\_blas\\_dcsr\\_rap4](#), [299](#)
  - [fasp\\_blas\\_dcsr\\_rap\\_agg](#), [300](#)
  - [fasp\\_blas\\_dcsr\\_rap\\_agg1](#), [301](#)
  - [fasp\\_blas\\_dcsr\\_vmv](#), [302](#)
- [BlaSpmvCSRL.c](#), [302](#)
  - [fasp\\_blas\\_dcsrl\\_mxv](#), [303](#)
- [BlaSpmvSTR.c](#), [303](#)
  - [fasp\\_blas\\_dstr\\_aAxy](#), [304](#)
  - [fasp\\_blas\\_dstr\\_mxv](#), [305](#)
  - [fasp\\_dstr\\_diagscale](#), [305](#)
- [BlaVector.c](#), [306](#)
  - [fasp\\_blas\\_dvec\\_axpy](#), [307](#)
  - [fasp\\_blas\\_dvec\\_axpyz](#), [307](#)
  - [fasp\\_blas\\_dvec\\_dotprod](#), [308](#)
  - [fasp\\_blas\\_dvec\\_norm1](#), [308](#)
  - [fasp\\_blas\\_dvec\\_norm2](#), [309](#)
  - [fasp\\_blas\\_dvec\\_norminf](#), [310](#)
  - [fasp\\_blas\\_dvec\\_reterr](#), [310](#)
- [block\\_dvector](#), [24](#)
  - [fasp\\_block.h](#), [325](#)
- [block\\_ivector](#), [25](#)
  - [fasp\\_block.h](#), [325](#)
- [CF\\_ORDER](#)
  - [fasp\\_const.h](#), [330](#)
- [CGPT](#)
  - [fasp\\_const.h](#), [331](#)
- [CLASSIC\\_AMG](#)
  - [fasp\\_const.h](#), [331](#)
- [COARSE\\_AC](#)
  - [fasp\\_const.h](#), [331](#)
- [COARSE\\_CR](#)
  - [fasp\\_const.h](#), [331](#)
- [COARSE\\_MIS](#)
  - [fasp\\_const.h](#), [331](#)
- [COARSE\\_RSP](#)
  - [fasp\\_const.h](#), [332](#)
- [COARSE\\_RS](#)
  - [fasp\\_const.h](#), [332](#)
- [CPFIRST](#)
  - [fasp\\_const.h](#), [332](#)
- [count](#)
  - [fasp.h](#), [322](#)
- [dBLCMat](#), [26](#)
  - [fasp\\_block.h](#), [325](#)
- [dBSRmat](#), [26](#)
  - [fasp\\_block.h](#), [326](#)
  - [JA](#), [27](#)
  - [val](#), [27](#)
- [dCOOmat](#), [27](#)
  - [fasp.h](#), [321](#)
- [dCSRLmat](#), [28](#)
  - [fasp.h](#), [321](#)
- [dCSRmat](#), [29](#)
  - [fasp.h](#), [321](#)
- [dCSRmat2SAMGInput](#)
  - [XtrSamg.c](#), [571](#)
- [DESCEND](#)
  - [fasp\\_const.h](#), [332](#)
- [DIAGONAL\\_PREF](#)
  - [fasp.h](#), [315](#)
- [DLMALLOC](#)
  - [fasp.h](#), [316](#)
- [dSTRmat](#), [31](#)
  - [fasp.h](#), [321](#)
- [ddenmat](#), [30](#)
  - [fasp.h](#), [321](#)
- [dlength](#)
  - [BlalO.c](#), [188](#)
- [doxygen.h](#), [312](#)
- [dvector](#), [32](#)
  - [fasp.h](#), [321](#)
- [dvector2SAMGInput](#)
  - [XtrSamg.c](#), [572](#)
- [e](#)
  - [grid2d](#), [33](#)
- [ERROR\\_ALLOC\\_MEM](#)
  - [fasp\\_const.h](#), [332](#)
- [ERROR\\_AMG\\_COARSE\\_TYPE](#)
  - [fasp\\_const.h](#), [333](#)
- [ERROR\\_AMG\\_COARSEING](#)
  - [fasp\\_const.h](#), [333](#)
- [ERROR\\_AMG\\_INTERP\\_TYPE](#)
  - [fasp\\_const.h](#), [333](#)
- [ERROR\\_AMG\\_SMOOTH\\_TYPE](#)
  - [fasp\\_const.h](#), [333](#)
- [ERROR\\_DATA\\_STRUCTURE](#)
  - [fasp\\_const.h](#), [333](#)
- [ERROR\\_DATA\\_ZERODIAG](#)
  - [fasp\\_const.h](#), [334](#)
- [ERROR\\_DUMMY\\_VAR](#)
  - [fasp\\_const.h](#), [334](#)
- [ERROR\\_INPUT\\_PAR](#)

- fasp\_const.h, [334](#)
- ERROR\_LIC\_TYPE
  - fasp\_const.h, [334](#)
- ERROR\_MAT\_SIZE
  - fasp\_const.h, [334](#)
- ERROR\_MISC
  - fasp\_const.h, [335](#)
- ERROR\_NUM\_BLOCKS
  - fasp\_const.h, [335](#)
- ERROR\_OPEN\_FILE
  - fasp\_const.h, [335](#)
- ERROR\_QUAD\_DIM
  - fasp\_const.h, [335](#)
- ERROR\_QUAD\_TYPE
  - fasp\_const.h, [335](#)
- ERROR\_REGRESS
  - fasp\_const.h, [336](#)
- ERROR\_SOLVER\_EXIT
  - fasp\_const.h, [336](#)
- ERROR\_SOLVER\_ILUSETUP
  - fasp\_const.h, [336](#)
- ERROR\_SOLVER\_MAXIT
  - fasp\_const.h, [336](#)
- ERROR\_SOLVER\_MISC
  - fasp\_const.h, [336](#)
- ERROR\_SOLVER\_PRECTYPE
  - fasp\_const.h, [337](#)
- ERROR\_SOLVER\_SOLSTAG
  - fasp\_const.h, [337](#)
- ERROR\_SOLVER\_STAG
  - fasp\_const.h, [337](#)
- ERROR\_SOLVER\_TOLSMALL
  - fasp\_const.h, [337](#)
- ERROR\_SOLVER\_TYPE
  - fasp\_const.h, [337](#)
- ERROR\_UNKNOWN
  - fasp\_const.h, [338](#)
- ERROR\_WRONG\_FILE
  - fasp\_const.h, [338](#)
- edges
  - grid2d, [33](#)
- ediri
  - grid2d, [33](#)
- efather
  - grid2d, [33](#)
- FALSE
  - fasp\_const.h, [338](#)
- FASP\_GSRB
  - fasp.h, [316](#)
- FASP\_SUCCESS
  - fasp\_const.h, [338](#)
- FASP\_VERSION
  - fasp.h, [316](#)
- FGPT
  - fasp\_const.h, [338](#)
- FPPFIRST
  - fasp\_const.h, [339](#)
- fasp.h, [312](#)
  - \_\_FASP\_HEADER\_\_, [315](#)
  - ABS, [315](#)
  - count, [322](#)
  - dCOOmat, [321](#)
  - dCSRmat, [321](#)
  - dCSRmat, [321](#)
  - DIAGONAL\_PREF, [315](#)
  - DLMALLOC, [316](#)
  - dSTRmat, [321](#)
  - ddenmat, [321](#)
  - dvector, [321](#)
  - FASP\_GSRB, [316](#)
  - FASP\_VERSION, [316](#)
  - GE, [316](#)
  - GT, [317](#)
  - iCOOmat, [321](#)
  - iCSRmat, [322](#)
  - IMAP, [322](#)
  - INT, [317](#)
  - ISNAN, [317](#)
  - idenmat, [322](#)
  - ivector, [322](#)
  - LONGLONG, [318](#)
  - LONG, [318](#)
  - LE, [317](#)
  - LS, [318](#)
  - MAXIMAP, [322](#)
  - MAX, [318](#)
  - MIN, [319](#)
  - NEDMALLOC, [319](#)
  - nx\_rb, [323](#)
  - ny\_rb, [323](#)
  - nz\_rb, [323](#)
  - PUT\_INT, [319](#)
  - PUT\_REAL, [319](#)
  - REAL, [320](#)
  - RS\_C1, [320](#)
  - SHORT, [320](#)
  - total\_alloc\_count, [323](#)
  - total\_alloc\_mem, [323](#)
- fasp\_BinarySearch
  - AuxSort.c, [117](#)
- fasp\_amg\_amli\_coef
  - PreMGRecurAMLI.c, [515](#)
- fasp\_amg\_coarsening\_cr
  - PreAMGCoarsenCR.c, [465](#)
- fasp\_amg\_coarsening\_rs
  - PreAMGCoarsenRS.c, [466](#)
- fasp\_amg\_data\_bsr\_create

PreDataInit.c, [504](#)  
fasp\_amg\_data\_bsr\_free  
  PreDataInit.c, [505](#)  
fasp\_amg\_data\_create  
  PreDataInit.c, [505](#)  
fasp\_amg\_data\_free  
  PreDataInit.c, [506](#)  
fasp\_amg\_interp  
  PreAMGInterp.c, [468](#)  
fasp\_amg\_interp\_em  
  PreAMGInterpEM.c, [470](#)  
fasp\_amg\_interp\_trunc  
  PreAMGInterp.c, [469](#)  
fasp\_amg\_setup\_cr  
  PreAMGSetupCR.c, [471](#)  
fasp\_amg\_setup\_rs  
  PreAMGSetupRS.c, [473](#)  
fasp\_amg\_setup\_sa  
  PreAMGSetupSA.c, [474](#)  
fasp\_amg\_setup\_sa\_bsr  
  PreAMGSetupSABSR.c, [475](#)  
fasp\_amg\_setup\_ua  
  PreAMGSetupUA.c, [477](#)  
fasp\_amg\_setup\_ua\_bsr  
  PreAMGSetupUABSR.c, [478](#)  
fasp\_amg\_solve  
  PreMGSolve.c, [519](#)  
fasp\_amg\_solve\_amli  
  PreMGSolve.c, [519](#)  
fasp\_amg\_solve\_nl\_amli  
  PreMGSolve.c, [520](#)  
fasp\_array\_cp  
  AuxArray.c, [76](#)  
fasp\_array\_cp\_nc3  
  AuxArray.c, [76](#)  
fasp\_array\_cp\_nc5  
  AuxArray.c, [77](#)  
fasp\_array\_cp\_nc7  
  AuxArray.c, [77](#)  
fasp\_array\_null  
  AuxArray.c, [78](#)  
fasp\_array\_set  
  AuxArray.c, [79](#)  
fasp\_aux\_bbyteToldouble  
  AuxConvert.c, [81](#)  
fasp\_aux\_change\_endian4  
  AuxConvert.c, [82](#)  
fasp\_aux\_change\_endian8  
  AuxConvert.c, [82](#)  
fasp\_aux\_dQuickSort  
  AuxSort.c, [111](#)  
fasp\_aux\_dQuickSortIndex  
  AuxSort.c, [112](#)  
fasp\_aux\_givens  
  AuxGivens.c, [83](#)  
fasp\_aux\_iQuickSort  
  AuxSort.c, [113](#)  
fasp\_aux\_iQuickSortIndex  
  AuxSort.c, [113](#)  
fasp\_aux\_merge  
  AuxSort.c, [115](#)  
fasp\_aux\_msort  
  AuxSort.c, [116](#)  
fasp\_aux\_unique  
  AuxSort.c, [116](#)  
fasp\_blas\_array\_ax  
  BlaArray.c, [133](#)  
fasp\_blas\_array\_axpby  
  BlaArray.c, [133](#)  
fasp\_blas\_array\_axpy  
  BlaArray.c, [134](#)  
fasp\_blas\_array\_axpy\_nc2  
  BlaSmallMat.c, [195](#)  
fasp\_blas\_array\_axpy\_nc3  
  BlaSmallMat.c, [195](#)  
fasp\_blas\_array\_axpy\_nc5  
  BlaSmallMat.c, [196](#)  
fasp\_blas\_array\_axpy\_nc7  
  BlaSmallMat.c, [197](#)  
fasp\_blas\_array\_axpyz  
  BlaArray.c, [135](#)  
fasp\_blas\_array\_axpyz\_nc2  
  BlaSmallMat.c, [197](#)  
fasp\_blas\_array\_axpyz\_nc3  
  BlaSmallMat.c, [198](#)  
fasp\_blas\_array\_axpyz\_nc5  
  BlaSmallMat.c, [199](#)  
fasp\_blas\_array\_axpyz\_nc7  
  BlaSmallMat.c, [199](#)  
fasp\_blas\_array\_dotprod  
  BlaArray.c, [136](#)  
fasp\_blas\_array\_norm1  
  BlaArray.c, [136](#)  
fasp\_blas\_array\_norm2  
  BlaArray.c, [137](#)  
fasp\_blas\_array\_norminf  
  BlaArray.c, [138](#)  
fasp\_blas\_dbldc\_aAxy  
  BlaSpmvBLC.c, [282](#)  
fasp\_blas\_dbldc\_mxv  
  BlaSpmvBLC.c, [283](#)  
fasp\_blas\_dbsr\_aAxyby  
  BlaSpmvBSR.c, [284](#)  
fasp\_blas\_dbsr\_aAxy  
  BlaSpmvBSR.c, [285](#)  
fasp\_blas\_dbsr\_aAxy\_agg  
  BlaSpmvBSR.c, [286](#)  
fasp\_blas\_dbsr\_axm

- BlaSpmvBSR.c, [287](#)
- fasp\_blas\_dbsr\_mxm
  - BlaSpmvBSR.c, [287](#)
- fasp\_blas\_dbsr\_m xv
  - BlaSpmvBSR.c, [288](#)
- fasp\_blas\_dbsr\_m xv\_agg
  - BlaSpmvBSR.c, [289](#)
- fasp\_blas\_dbsr\_rap
  - BlaSpmvBSR.c, [289](#)
- fasp\_blas\_dbsr\_rap1
  - BlaSpmvBSR.c, [290](#)
- fasp\_blas\_dbsr\_rap\_agg
  - BlaSpmvBSR.c, [291](#)
- fasp\_blas\_dcsr\_aA xpy
  - BlaSpmvCSR.c, [293](#)
- fasp\_blas\_dcsr\_aA xpy\_agg
  - BlaSpmvCSR.c, [293](#)
- fasp\_blas\_dcsr\_add
  - BlaSpmvCSR.c, [294](#)
- fasp\_blas\_dcsr\_axm
  - BlaSpmvCSR.c, [295](#)
- fasp\_blas\_dcsr\_m xm
  - BlaSpmvCSR.c, [295](#)
- fasp\_blas\_dcsr\_m xv
  - BlaSpmvCSR.c, [296](#)
- fasp\_blas\_dcsr\_m xv\_agg
  - BlaSpmvCSR.c, [297](#)
- fasp\_blas\_dcsr\_ptap
  - BlaSpmvCSR.c, [297](#)
- fasp\_blas\_dcsr\_rap
  - BlaSpmvCSR.c, [298](#)
- fasp\_blas\_dcsr\_rap2
  - BlaSpmvCSR.c, [299](#)
- fasp\_blas\_dcsr\_rap4
  - BlaSpmvCSR.c, [299](#)
- fasp\_blas\_dcsr\_rap\_agg
  - BlaSpmvCSR.c, [300](#)
- fasp\_blas\_dcsr\_rap\_agg1
  - BlaSpmvCSR.c, [301](#)
- fasp\_blas\_dcsr\_v mv
  - BlaSpmvCSR.c, [302](#)
- fasp\_blas\_dcsrl\_m xv
  - BlaSpmvCSRL.c, [303](#)
- fasp\_blas\_dstr\_aA xpy
  - BlaSpmvSTR.c, [304](#)
- fasp\_blas\_dstr\_m xv
  - BlaSpmvSTR.c, [305](#)
- fasp\_blas\_dvec\_axpy
  - BlaVector.c, [307](#)
- fasp\_blas\_dvec\_axpyz
  - BlaVector.c, [307](#)
- fasp\_blas\_dvec\_dotprod
  - BlaVector.c, [308](#)
- fasp\_blas\_dvec\_norm1
  - BlaVector.c, [308](#)
- fasp\_blas\_dvec\_norm2
  - BlaVector.c, [309](#)
- fasp\_blas\_dvec\_norminf
  - BlaVector.c, [310](#)
- fasp\_blas\_dvec\_relerr
  - BlaVector.c, [310](#)
- fasp\_blas\_smat\_aA xpy
  - BlaSmallMat.c, [200](#)
- fasp\_blas\_smat\_add
  - BlaSmallMat.c, [201](#)
- fasp\_blas\_smat\_axm
  - BlaSmallMat.c, [202](#)
- fasp\_blas\_smat\_mul
  - BlaSmallMat.c, [202](#)
- fasp\_blas\_smat\_mul\_nc2
  - BlaSmallMat.c, [203](#)
- fasp\_blas\_smat\_mul\_nc3
  - BlaSmallMat.c, [203](#)
- fasp\_blas\_smat\_mul\_nc5
  - BlaSmallMat.c, [204](#)
- fasp\_blas\_smat\_mul\_nc7
  - BlaSmallMat.c, [205](#)
- fasp\_blas\_smat\_m xv
  - BlaSmallMat.c, [205](#)
- fasp\_blas\_smat\_m xv\_nc2
  - BlaSmallMat.c, [206](#)
- fasp\_blas\_smat\_m xv\_nc3
  - BlaSmallMat.c, [206](#)
- fasp\_blas\_smat\_m xv\_nc5
  - BlaSmallMat.c, [207](#)
- fasp\_blas\_smat\_m xv\_nc7
  - BlaSmallMat.c, [208](#)
- fasp\_blas\_smat\_y mA x
  - BlaSmallMat.c, [208](#)
- fasp\_blas\_smat\_y mA x\_nc2
  - BlaSmallMat.c, [209](#)
- fasp\_blas\_smat\_y mA x\_nc3
  - BlaSmallMat.c, [210](#)
- fasp\_blas\_smat\_y mA x\_nc5
  - BlaSmallMat.c, [210](#)
- fasp\_blas\_smat\_y mA x\_nc7
  - BlaSmallMat.c, [211](#)
- fasp\_blas\_smat\_y pA x
  - BlaSmallMat.c, [212](#)
- fasp\_blas\_smat\_y pA x\_nc2
  - BlaSmallMat.c, [212](#)
- fasp\_blas\_smat\_y pA x\_nc3
  - BlaSmallMat.c, [213](#)
- fasp\_blas\_smat\_y pA x\_nc5
  - BlaSmallMat.c, [213](#)
- fasp\_blas\_smat\_y pA x\_nc7
  - BlaSmallMat.c, [214](#)
- fasp\_block.h, [324](#)

- \_\_FASPBLOCK\_HEADER\_\_, 325
  - block\_dvector, 325
  - block\_ivector, 325
  - dBLCmat, 325
  - dBSRmat, 326
  - iBLCmat, 326
- fasp\_check\_dCSRmat
  - BlaSparseCheck.c, 240
- fasp\_check\_diagdom
  - BlaSparseCheck.c, 241
- fasp\_check\_diagpos
  - BlaSparseCheck.c, 241
- fasp\_check\_diagzero
  - BlaSparseCheck.c, 242
- fasp\_check\_iCSRmat
  - BlaSparseCheck.c, 243
- fasp\_check\_symm
  - BlaSparseCheck.c, 243
- fasp\_chkerr
  - AuxMessage.c, 96
- fasp\_const.h, 326
  - AMLI\_CYCLE, 330
  - ASCEND, 330
  - BIGREAL, 330
  - CF\_ORDER, 330
  - CGPT, 331
  - CLASSIC\_AMG, 331
  - COARSE\_AC, 331
  - COARSE\_CR, 331
  - COARSE\_MIS, 331
  - COARSE\_RSP, 332
  - COARSE\_RS, 332
  - CPFIRST, 332
  - DESCEND, 332
  - ERROR\_ALLOC\_MEM, 332
  - ERROR\_AMG\_COARSE\_TYPE, 333
  - ERROR\_AMG\_COARSEING, 333
  - ERROR\_AMG\_INTERP\_TYPE, 333
  - ERROR\_AMG\_SMOOTH\_TYPE, 333
  - ERROR\_DATA\_STRUCTURE, 333
  - ERROR\_DATA\_ZERODIAG, 334
  - ERROR\_DUMMY\_VAR, 334
  - ERROR\_INPUT\_PAR, 334
  - ERROR\_LIC\_TYPE, 334
  - ERROR\_MAT\_SIZE, 334
  - ERROR\_MISC, 335
  - ERROR\_NUM\_BLOCKS, 335
  - ERROR\_OPEN\_FILE, 335
  - ERROR\_QUAD\_DIM, 335
  - ERROR\_QUAD\_TYPE, 335
  - ERROR\_REGRESS, 336
  - ERROR\_SOLVER\_EXIT, 336
  - ERROR\_SOLVER\_ILUSETUP, 336
  - ERROR\_SOLVER\_MAXIT, 336
  - ERROR\_SOLVER\_MISC, 336
  - ERROR\_SOLVER\_PRECTYPE, 337
  - ERROR\_SOLVER\_SOLSTAG, 337
  - ERROR\_SOLVER\_STAG, 337
  - ERROR\_SOLVER\_TOLSMALL, 337
  - ERROR\_SOLVER\_TYPE, 337
  - ERROR\_UNKNOWN, 338
  - ERROR\_WRONG\_FILE, 338
  - FALSE, 338
  - FASP\_SUCCESS, 338
  - FGPT, 338
  - FPPFIRST, 339
  - G0PT, 339
  - ILU\_MC\_OMP, 339
  - ILUk, 339
  - ILUt, 340
  - ILUtp, 340
  - INTERP\_DIR, 340
  - INTERP\_ENG, 340
  - INTERP\_EXT, 341
  - INTERP\_STD, 341
  - ISPT, 341
  - MAT\_BLC, 342
  - MAT\_BSR, 342
  - MAT\_CSRL, 342
  - MAT\_CSR, 342
  - MAT\_FREE, 343
  - MAT\_STR, 343
  - MAT\_SymCSR, 343
  - MAT\_bBSR, 341
  - MAT\_bCSR, 341
  - MAT\_bSTR, 342
  - MAX\_AMG\_LVL, 343
  - MAX\_CRATE, 343
  - MAX\_REFINE\_LVL, 344
  - MAX\_RESTART, 344
  - MAX\_STAG, 344
  - MIN\_CDOF, 344
  - MIN\_CRATE, 344
  - NL\_AMLI\_CYCLE, 345
  - NO\_ORDER, 345
  - OFF, 345
  - OPENMP\_HOLDS, 346
  - ON, 345
  - PAIRWISE, 346
  - PREC\_AMG, 346
  - PREC\_DIAG, 346
  - PREC\_FMG, 347
  - PREC\_ILU, 347
  - PREC\_NULL, 347
  - PREC\_SCHWARZ, 347
  - PRINT\_ALL, 347
  - PRINT\_MIN, 348
  - PRINT\_MORE, 348

PRINT\_MOST, [348](#)  
 PRINT\_NONE, [348](#)  
 PRINT\_SOME, [348](#)  
 SA\_AMG, [349](#)  
 SCHWARZ\_BACKWARD, [349](#)  
 SCHWARZ\_FORWARD, [349](#)  
 SCHWARZ\_SYMMETRIC, [349](#)  
 SMALLREAL2, [350](#)  
 SMALLREAL, [349](#)  
 SMOOTHER\_BLKOil, [350](#)  
 SMOOTHER\_CG, [350](#)  
 SMOOTHER\_GSOR, [350](#)  
 SMOOTHER\_GS, [350](#)  
 SMOOTHER\_JACOBI, [351](#)  
 SMOOTHER\_L1DIAG, [351](#)  
 SMOOTHER\_POLY, [351](#)  
 SMOOTHER\_SGSOR, [351](#)  
 SMOOTHER\_SGS, [351](#)  
 SMOOTHER\_SOR, [352](#)  
 SMOOTHER\_SPETEN, [352](#)  
 SMOOTHER\_SSOR, [352](#)  
 SOLVER\_AMG, [352](#)  
 SOLVER\_BiCGstab, [352](#)  
 SOLVER\_CG, [353](#)  
 SOLVER\_DEFAULT, [353](#)  
 SOLVER\_FMG, [353](#)  
 SOLVER\_GCG, [353](#)  
 SOLVER\_GCR, [353](#)  
 SOLVER\_GMRES, [354](#)  
 SOLVER\_MUMPS, [354](#)  
 SOLVER\_MinRes, [354](#)  
 SOLVER\_PARDISO, [354](#)  
 SOLVER\_SBiCGstab, [354](#)  
 SOLVER\_SCG, [355](#)  
 SOLVER\_SGCG, [355](#)  
 SOLVER\_SGMRES, [355](#)  
 SOLVER\_SMinRes, [355](#)  
 SOLVER\_SUPERLU, [355](#)  
 SOLVER\_SVFGMRES, [356](#)  
 SOLVER\_SVGMRES, [356](#)  
 SOLVER\_UMFPACK, [356](#)  
 SOLVER\_VBiCGstab, [356](#)  
 SOLVER\_VFGMRES, [356](#)  
 SOLVER\_VGMRES, [357](#)  
 STAG\_RATIO, [357](#)  
 STOP\_MOD\_REL\_RES, [357](#)  
 STOP\_REL\_PRECRES, [357](#)  
 STOP\_REL\_RES, [357](#)  
 TRUE, [358](#)  
 UA\_AMG, [358](#)  
 UNPT, [358](#)  
 USERDEFINED, [358](#)  
 V\_CYCLE, [359](#)  
 VMB, [359](#)  
 W\_CYCLE, [359](#)  
 fasp\_dblc\_free  
     BlaSparseBLC.c, [227](#)  
 fasp\_dbsr\_alloc  
     BlaSparseBSR.c, [229](#)  
 fasp\_dbsr\_cp  
     BlaSparseBSR.c, [230](#)  
 fasp\_dbsr\_create  
     BlaSparseBSR.c, [230](#)  
 fasp\_dbsr\_diagLU2  
     BlaSparseBSR.c, [234](#)  
 fasp\_dbsr\_diagLU  
     BlaSparseBSR.c, [234](#)  
 fasp\_dbsr\_diaginv  
     BlaSparseBSR.c, [231](#)  
 fasp\_dbsr\_diaginv2  
     BlaSparseBSR.c, [232](#)  
 fasp\_dbsr\_diaginv3  
     BlaSparseBSR.c, [232](#)  
 fasp\_dbsr\_diaginv4  
     BlaSparseBSR.c, [233](#)  
 fasp\_dbsr\_diagpref  
     BlaSparseBSR.c, [235](#)  
 fasp\_dbsr\_free  
     BlaSparseBSR.c, [236](#)  
 fasp\_dbsr\_getblk  
     BlaSparseBLC.c, [227](#)  
 fasp\_dbsr\_getdiag  
     BlaSparseBSR.c, [236](#)  
 fasp\_dbsr\_getdiaginv  
     BlaSparseBSR.c, [237](#)  
 fasp\_dbsr\_null  
     BlaSparseBSR.c, [238](#)  
 fasp\_dbsr\_perm  
     BlaSparseBSR.c, [238](#)  
 fasp\_dbsr\_plot  
     AuxGraphics.c, [85](#)  
 fasp\_dbsr\_print  
     BlalO.c, [163](#)  
 fasp\_dbsr\_read  
     BlalO.c, [164](#)  
 fasp\_dbsr\_subplot  
     AuxGraphics.c, [85](#)  
 fasp\_dbsr\_trans  
     BlaSparseBSR.c, [239](#)  
 fasp\_dbsr\_write  
     BlalO.c, [165](#)  
 fasp\_dbsr\_write\_coo  
     BlalO.c, [165](#)  
 fasp\_dcoo1\_read  
     BlalO.c, [166](#)  
 fasp\_dcoo\_alloc  
     BlaSparseCOO.c, [245](#)  
 fasp\_dcoo\_create



BlaSparseCOO.c, 245  
fasp\_dcoo\_free  
    BlaSparseCOO.c, 246  
fasp\_dcoo\_print  
    BlaIO.c, 167  
fasp\_dcoo\_read  
    BlaIO.c, 167  
fasp\_dcoo\_shift  
    BlaSparseCOO.c, 246  
fasp\_dcoo\_shift\_read  
    BlaIO.c, 168  
fasp\_dcoo\_write  
    BlaIO.c, 169  
fasp\_dcsr\_CMK\_order  
    BlaOrderingCSR.c, 189  
fasp\_dcsr\_RCMK\_order  
    BlaOrderingCSR.c, 190  
fasp\_dcsr\_alloc  
    BlaSparseCSR.c, 249  
fasp\_dcsr\_bandwidth  
    BlaSparseCSR.c, 249  
fasp\_dcsr\_compress  
    BlaSparseCSR.c, 250  
fasp\_dcsr\_compress\_inplace  
    BlaSparseCSR.c, 250  
fasp\_dcsr\_cp  
    BlaSparseCSR.c, 251  
fasp\_dcsr\_create  
    BlaSparseCSR.c, 252  
fasp\_dcsr\_diagpref  
    BlaSparseCSR.c, 252  
fasp\_dcsr\_eig  
    BlaEigen.c, 139  
fasp\_dcsr\_free  
    BlaSparseCSR.c, 253  
fasp\_dcsr\_getblk  
    BlaSparseCSR.c, 254  
fasp\_dcsr\_getcol  
    BlaSparseCSR.c, 254  
fasp\_dcsr\_getdiag  
    BlaSparseCSR.c, 255  
fasp\_dcsr\_multicoloring  
    BlaSparseCSR.c, 256  
fasp\_dcsr\_null  
    BlaSparseCSR.c, 256  
fasp\_dcsr\_perm  
    BlaSparseCSR.c, 257  
fasp\_dcsr\_permz  
    BlaSparseCSR.c, 257  
fasp\_dcsr\_plot  
    AuxGraphics.c, 86  
fasp\_dcsr\_print  
    BlaIO.c, 169  
fasp\_dcsr\_read  
    BlaIO.c, 170  
fasp\_dcsr\_regdiag  
    BlaSparseCSR.c, 258  
fasp\_dcsr\_schwarz\_backward\_smoother  
    BlaSchwarzSetup.c, 191  
fasp\_dcsr\_schwarz\_forward\_smoother  
    BlaSchwarzSetup.c, 192  
fasp\_dcsr\_shift  
    BlaSparseCSR.c, 259  
fasp\_dcsr\_sort  
    BlaSparseCSR.c, 259  
fasp\_dcsr\_sortz  
    BlaSparseCSR.c, 260  
fasp\_dcsr\_subplot  
    AuxGraphics.c, 87  
fasp\_dcsr\_symdiagscale  
    BlaSparseCSR.c, 260  
fasp\_dcsr\_sympart  
    BlaSparseCSR.c, 261  
fasp\_dcsr\_trans  
    BlaSparseCSR.c, 262  
fasp\_dcsr\_transz  
    BlaSparseCSR.c, 262  
fasp\_dcsr\_write\_coo  
    BlaIO.c, 170  
fasp\_dcsr\_create  
    BlaSparseCSR.c, 266  
fasp\_dcsr\_free  
    BlaSparseCSR.c, 267  
fasp\_dcsrvec1\_read  
    BlaIO.c, 171  
fasp\_dcsrvec1\_write  
    BlaIO.c, 172  
fasp\_dcsrvec2\_read  
    BlaIO.c, 173  
fasp\_dcsrvec2\_write  
    BlaIO.c, 174  
fasp\_dmtx\_read  
    BlaIO.c, 174  
fasp\_dmtxsym\_read  
    BlaIO.c, 175  
fasp\_dstr\_alloc  
    BlaSparseSTR.c, 268  
fasp\_dstr\_cp  
    BlaSparseSTR.c, 269  
fasp\_dstr\_create  
    BlaSparseSTR.c, 269  
fasp\_dstr\_diagscale  
    BlaSpmvSTR.c, 305  
fasp\_dstr\_free  
    BlaSparseSTR.c, 270  
fasp\_dstr\_null  
    BlaSparseSTR.c, 270  
fasp\_dstr\_print

- BlaIO.c, 176
- fasp\_dstr\_read
  - BlaIO.c, 176
- fasp\_dstr\_write
  - BlaIO.c, 177
- fasp\_dvec\_alloc
  - AuxVector.c, 124
- fasp\_dvec\_cp
  - AuxVector.c, 124
- fasp\_dvec\_create
  - AuxVector.c, 125
- fasp\_dvec\_free
  - AuxVector.c, 125
- fasp\_dvec\_isnan
  - AuxVector.c, 126
- fasp\_dvec\_maxdiff
  - AuxVector.c, 126
- fasp\_dvec\_null
  - AuxVector.c, 127
- fasp\_dvec\_print
  - BlaIO.c, 178
- fasp\_dvec\_rand
  - AuxVector.c, 128
- fasp\_dvec\_read
  - BlaIO.c, 178
- fasp\_dvec\_set
  - AuxVector.c, 128
- fasp\_dvec\_symdiagscale
  - AuxVector.c, 129
- fasp\_dvec\_write
  - BlaIO.c, 179
- fasp\_dvecind\_read
  - BlaIO.c, 179
- fasp\_dvecind\_write
  - BlaIO.c, 180
- fasp\_famg\_solve
  - PreMGSolve.c, 521
- fasp\_format\_dblc\_dcsr
  - BlaFormat.c, 141
- fasp\_format\_dbsr\_dcoo
  - BlaFormat.c, 141
- fasp\_format\_dbsr\_dcsr
  - BlaFormat.c, 142
- fasp\_format\_dcoo\_dcsr
  - BlaFormat.c, 143
- fasp\_format\_dcsr\_dbsr
  - BlaFormat.c, 143
- fasp\_format\_dcsr\_dcoo
  - BlaFormat.c, 144
- fasp\_format\_dcsrl\_dcsr
  - BlaFormat.c, 145
- fasp\_format\_dstr\_dbsr
  - BlaFormat.c, 145
- fasp\_format\_dstr\_dcsr
  - BlaFormat.c, 146
- fasp\_fwrapper\_amg\_
  - SolWrapper.c, 564
- fasp\_fwrapper\_krylov\_amg\_
  - SolWrapper.c, 564
- fasp\_generate\_diaginv\_block
  - ltrSmootherSTR.c, 388
- fasp\_get\_start\_end
  - AuxThreads.c, 120
- fasp\_gettime
  - AuxTiming.c, 122
- fasp\_grid.h, 359
  - \_\_FASPGRID\_HEADER\_\_, 360
  - grid2d, 360
  - pcgrid2d, 360
  - pgrid2d, 361
- fasp\_grid2d\_plot
  - AuxGraphics.c, 87
- fasp\_hb\_read
  - BlaIO.c, 181
- fasp\_iarray\_cp
  - AuxArray.c, 79
- fasp\_iarray\_set
  - AuxArray.c, 80
- fasp\_icsr\_cp
  - BlaSparseCSR.c, 263
- fasp\_icsr\_create
  - BlaSparseCSR.c, 263
- fasp\_icsr\_free
  - BlaSparseCSR.c, 264
- fasp\_icsr\_null
  - BlaSparseCSR.c, 265
- fasp\_icsr\_trans
  - BlaSparseCSR.c, 265
- fasp\_ilu\_data\_create
  - PreDataInit.c, 506
- fasp\_ilu\_data\_free
  - PreDataInit.c, 507
- fasp\_ilu\_data\_null
  - PreDataInit.c, 507
- fasp\_ilu\_dbsr\_setup
  - BlaLUSetupBSR.c, 155
- fasp\_ilu\_dbsr\_setup\_levsch\_omp
  - BlaLUSetupBSR.c, 156
- fasp\_ilu\_dbsr\_setup\_mc\_omp
  - BlaLUSetupBSR.c, 156
- fasp\_ilu\_dbsr\_setup\_omp
  - BlaLUSetupBSR.c, 157
- fasp\_ilu\_dcsr\_setup
  - BlaLUSetupCSR.c, 159
- fasp\_ilu\_dstr\_setup0
  - BlaLUSetupSTR.c, 160
- fasp\_ilu\_dstr\_setup1
  - BlaLUSetupSTR.c, 160

fasp\_iluk  
  BlalLU.c, [147](#)

fasp\_ilut  
  BlalLU.c, [149](#)

fasp\_ilutp  
  BlalLU.c, [150](#)

fasp\_ivec\_alloc  
  AuxVector.c, [130](#)

fasp\_ivec\_create  
  AuxVector.c, [130](#)

fasp\_ivec\_free  
  AuxVector.c, [131](#)

fasp\_ivec\_print  
  BlalO.c, [181](#)

fasp\_ivec\_read  
  BlalO.c, [182](#)

fasp\_ivec\_set  
  AuxVector.c, [131](#)

fasp\_ivec\_write  
  BlalO.c, [183](#)

fasp\_ivecind\_read  
  BlalO.c, [183](#)

fasp\_matrix\_read  
  BlalO.c, [184](#)

fasp\_matrix\_read\_bin  
  BlalO.c, [185](#)

fasp\_matrix\_write  
  BlalO.c, [185](#)

fasp\_mem\_calloc  
  AuxMemory.c, [91](#)

fasp\_mem\_check  
  AuxMemory.c, [91](#)

fasp\_mem\_free  
  AuxMemory.c, [92](#)

fasp\_mem\_iludata\_check  
  AuxMemory.c, [93](#)

fasp\_mem\_realloc  
  AuxMemory.c, [93](#)

fasp\_mem\_usage  
  AuxMemory.c, [94](#)

fasp\_multicolors\_independent\_set  
  AuxSort.c, [118](#)

fasp\_param\_amg\_init  
  AuxParam.c, [101](#)

fasp\_param\_amg\_print  
  AuxParam.c, [101](#)

fasp\_param\_amg\_set  
  AuxParam.c, [102](#)

fasp\_param\_amg\_to\_prec  
  AuxParam.c, [102](#)

fasp\_param\_amg\_to\_prec\_bsr  
  AuxParam.c, [103](#)

fasp\_param\_check  
  AuxInput.c, [89](#)

fasp\_param\_ilu\_init  
  AuxParam.c, [103](#)

fasp\_param\_ilu\_print  
  AuxParam.c, [104](#)

fasp\_param\_ilu\_set  
  AuxParam.c, [104](#)

fasp\_param\_init  
  AuxParam.c, [105](#)

fasp\_param\_input  
  AuxInput.c, [89](#)

fasp\_param\_input\_init  
  AuxParam.c, [105](#)

fasp\_param\_prec\_to\_amg  
  AuxParam.c, [106](#)

fasp\_param\_prec\_to\_amg\_bsr  
  AuxParam.c, [106](#)

fasp\_param\_schwarz\_init  
  AuxParam.c, [107](#)

fasp\_param\_schwarz\_print  
  AuxParam.c, [107](#)

fasp\_param\_schwarz\_set  
  AuxParam.c, [108](#)

fasp\_param\_set  
  AuxParam.c, [108](#)

fasp\_param\_solver\_init  
  AuxParam.c, [109](#)

fasp\_param\_solver\_print  
  AuxParam.c, [109](#)

fasp\_param\_solver\_set  
  AuxParam.c, [110](#)

fasp\_poisson\_fgmg1d  
  SolGMGPoisson.c, [549](#)

fasp\_poisson\_fgmg2d  
  SolGMGPoisson.c, [549](#)

fasp\_poisson\_fgmg3d  
  SolGMGPoisson.c, [550](#)

fasp\_poisson\_gmg1d  
  SolGMGPoisson.c, [551](#)

fasp\_poisson\_gmg2d  
  SolGMGPoisson.c, [551](#)

fasp\_poisson\_gmg3d  
  SolGMGPoisson.c, [552](#)

fasp\_poisson\_gmgcg1d  
  SolGMGPoisson.c, [553](#)

fasp\_poisson\_gmgcg2d  
  SolGMGPoisson.c, [554](#)

fasp\_poisson\_gmgcg3d  
  SolGMGPoisson.c, [555](#)

fasp\_precond\_amg  
  PreCSR.c, [495](#)

fasp\_precond\_amg\_nk  
  PreCSR.c, [496](#)

fasp\_precond\_amli  
  PreCSR.c, [496](#)

fasp\_precond\_block\_SGS\_3  
     PreBLC.c, [483](#)  
 fasp\_precond\_block\_SGS\_3\_amg  
     PreBLC.c, [484](#)  
 fasp\_precond\_block\_diag\_3  
     PreBLC.c, [480](#)  
 fasp\_precond\_block\_diag\_3\_amg  
     PreBLC.c, [480](#)  
 fasp\_precond\_block\_diag\_4  
     PreBLC.c, [481](#)  
 fasp\_precond\_block\_lower\_3  
     PreBLC.c, [481](#)  
 fasp\_precond\_block\_lower\_3\_amg  
     PreBLC.c, [482](#)  
 fasp\_precond\_block\_lower\_4  
     PreBLC.c, [482](#)  
 fasp\_precond\_block\_upper\_3  
     PreBLC.c, [484](#)  
 fasp\_precond\_block\_upper\_3\_amg  
     PreBLC.c, [485](#)  
 fasp\_precond\_data\_null  
     PreDataInit.c, [508](#)  
 fasp\_precond\_dbsr\_amg  
     PreBSR.c, [487](#)  
 fasp\_precond\_dbsr\_amg\_nk  
     PreBSR.c, [487](#)  
 fasp\_precond\_dbsr\_diag  
     PreBSR.c, [488](#)  
 fasp\_precond\_dbsr\_diag\_nc2  
     PreBSR.c, [489](#)  
 fasp\_precond\_dbsr\_diag\_nc3  
     PreBSR.c, [489](#)  
 fasp\_precond\_dbsr\_diag\_nc5  
     PreBSR.c, [490](#)  
 fasp\_precond\_dbsr\_diag\_nc7  
     PreBSR.c, [491](#)  
 fasp\_precond\_dbsr\_ilu  
     PreBSR.c, [492](#)  
 fasp\_precond\_dbsr\_ilu\_ls\_omp  
     PreBSR.c, [492](#)  
 fasp\_precond\_dbsr\_ilu\_mc\_omp  
     PreBSR.c, [493](#)  
 fasp\_precond\_dbsr\_nl\_amli  
     PreBSR.c, [494](#)  
 fasp\_precond\_diag  
     PreCSR.c, [497](#)  
 fasp\_precond\_dstr\_blockgs  
     PreSTR.c, [522](#)  
 fasp\_precond\_dstr\_diag  
     PreSTR.c, [523](#)  
 fasp\_precond\_dstr\_ilu0  
     PreSTR.c, [523](#)  
 fasp\_precond\_dstr\_ilu0\_backward  
     PreSTR.c, [524](#)  
 fasp\_precond\_dstr\_ilu0\_forward  
     PreSTR.c, [524](#)  
 fasp\_precond\_dstr\_ilu1  
     PreSTR.c, [525](#)  
 fasp\_precond\_dstr\_ilu1\_backward  
     PreSTR.c, [526](#)  
 fasp\_precond\_dstr\_ilu1\_forward  
     PreSTR.c, [526](#)  
 fasp\_precond\_famg  
     PreCSR.c, [498](#)  
 fasp\_precond\_free  
     PreCSR.c, [498](#)  
 fasp\_precond\_ilu  
     PreCSR.c, [499](#)  
 fasp\_precond\_ilu\_backward  
     PreCSR.c, [499](#)  
 fasp\_precond\_ilu\_forward  
     PreCSR.c, [501](#)  
 fasp\_precond\_nl\_amli  
     PreCSR.c, [501](#)  
 fasp\_precond\_null  
     PreDataInit.c, [508](#)  
 fasp\_precond\_schwarz  
     PreCSR.c, [502](#)  
 fasp\_precond\_setup  
     PreCSR.c, [503](#)  
 fasp\_precond\_sweeping  
     PreBLC.c, [485](#)  
 fasp\_schwarz\_data\_free  
     PreDataInit.c, [509](#)  
 fasp\_schwarz\_setup  
     BlaSchwarzSetup.c, [192](#)  
 fasp\_set\_GS\_threads  
     AuxThreads.c, [120](#)  
 fasp\_smat\_Linfinity  
     BlaSmallMatInv.c, [223](#)  
 fasp\_smat\_identity  
     BlaSmallMatInv.c, [216](#)  
 fasp\_smat\_identity\_nc2  
     BlaSmallMatInv.c, [217](#)  
 fasp\_smat\_identity\_nc3  
     BlaSmallMatInv.c, [217](#)  
 fasp\_smat\_identity\_nc5  
     BlaSmallMatInv.c, [218](#)  
 fasp\_smat\_identity\_nc7  
     BlaSmallMatInv.c, [218](#)  
 fasp\_smat\_inv  
     BlaSmallMatInv.c, [219](#)  
 fasp\_smat\_inv\_nc  
     BlaSmallMatInv.c, [219](#)  
 fasp\_smat\_inv\_nc2  
     BlaSmallMatInv.c, [220](#)  
 fasp\_smat\_inv\_nc3  
     BlaSmallMatInv.c, [220](#)

- fasp\_smat\_inv\_nc4
  - BlaSmallMatInv.c, [221](#)
- fasp\_smat\_inv\_nc5
  - BlaSmallMatInv.c, [221](#)
- fasp\_smat\_inv\_nc7
  - BlaSmallMatInv.c, [222](#)
- fasp\_smat\_invp\_nc
  - BlaSmallMatInv.c, [222](#)
- fasp\_smat\_lu\_decomp
  - BlaSmallMatLU.c, [224](#)
- fasp\_smat\_lu\_solve
  - BlaSmallMatLU.c, [225](#)
- fasp\_smoother\_dbsr\_gs
  - ltrSmootherBSR.c, [362](#)
- fasp\_smoother\_dbsr\_gs1
  - ltrSmootherBSR.c, [363](#)
- fasp\_smoother\_dbsr\_gs\_ascend
  - ltrSmootherBSR.c, [364](#)
- fasp\_smoother\_dbsr\_gs\_ascend1
  - ltrSmootherBSR.c, [364](#)
- fasp\_smoother\_dbsr\_gs\_descend
  - ltrSmootherBSR.c, [365](#)
- fasp\_smoother\_dbsr\_gs\_descend1
  - ltrSmootherBSR.c, [366](#)
- fasp\_smoother\_dbsr\_gs\_order1
  - ltrSmootherBSR.c, [366](#)
- fasp\_smoother\_dbsr\_gs\_order2
  - ltrSmootherBSR.c, [367](#)
- fasp\_smoother\_dbsr\_ilu
  - ltrSmootherBSR.c, [368](#)
- fasp\_smoother\_dbsr\_jacobi
  - ltrSmootherBSR.c, [368](#)
- fasp\_smoother\_dbsr\_jacobi1
  - ltrSmootherBSR.c, [369](#)
- fasp\_smoother\_dbsr\_jacobi\_setup
  - ltrSmootherBSR.c, [369](#)
- fasp\_smoother\_dbsr\_sor
  - ltrSmootherBSR.c, [370](#)
- fasp\_smoother\_dbsr\_sor1
  - ltrSmootherBSR.c, [371](#)
- fasp\_smoother\_dbsr\_sor\_ascend
  - ltrSmootherBSR.c, [371](#)
- fasp\_smoother\_dbsr\_sor\_descend
  - ltrSmootherBSR.c, [372](#)
- fasp\_smoother\_dbsr\_sor\_order
  - ltrSmootherBSR.c, [373](#)
- fasp\_smoother\_dcsr\_L1diag
  - ltrSmootherCSR.c, [380](#)
- fasp\_smoother\_dcsr\_gs
  - ltrSmootherCSR.c, [375](#)
- fasp\_smoother\_dcsr\_gs\_cf
  - ltrSmootherCSR.c, [376](#)
- fasp\_smoother\_dcsr\_gs3d
  - ltrSmootherCSR.c, [376](#)
- fasp\_smoother\_dcsr\_gscr
  - ltrSmootherCSRcr.c, [383](#)
- fasp\_smoother\_dcsr\_ilu
  - ltrSmootherCSR.c, [377](#)
- fasp\_smoother\_dcsr\_jacobi
  - ltrSmootherCSR.c, [378](#)
- fasp\_smoother\_dcsr\_kaczmarz
  - ltrSmootherCSR.c, [379](#)
- fasp\_smoother\_dcsr\_poly
  - ltrSmootherCSRpoly.c, [385](#)
- fasp\_smoother\_dcsr\_poly\_old
  - ltrSmootherCSRpoly.c, [385](#)
- fasp\_smoother\_dcsr\_sgs
  - ltrSmootherCSR.c, [380](#)
- fasp\_smoother\_dcsr\_sor
  - ltrSmootherCSR.c, [381](#)
- fasp\_smoother\_dcsr\_sor\_cf
  - ltrSmootherCSR.c, [382](#)
- fasp\_smoother\_dstr\_gs
  - ltrSmootherSTR.c, [388](#)
- fasp\_smoother\_dstr\_gs1
  - ltrSmootherSTR.c, [389](#)
- fasp\_smoother\_dstr\_gs\_ascend
  - ltrSmootherSTR.c, [390](#)
- fasp\_smoother\_dstr\_gs\_cf
  - ltrSmootherSTR.c, [390](#)
- fasp\_smoother\_dstr\_gs\_descend
  - ltrSmootherSTR.c, [392](#)
- fasp\_smoother\_dstr\_gs\_order
  - ltrSmootherSTR.c, [393](#)
- fasp\_smoother\_dstr\_jacobi
  - ltrSmootherSTR.c, [393](#)
- fasp\_smoother\_dstr\_jacobi1
  - ltrSmootherSTR.c, [394](#)
- fasp\_smoother\_dstr\_schwarz
  - ltrSmootherSTR.c, [394](#)
- fasp\_smoother\_dstr\_sor
  - ltrSmootherSTR.c, [395](#)
- fasp\_smoother\_dstr\_sor1
  - ltrSmootherSTR.c, [396](#)
- fasp\_smoother\_dstr\_sor\_ascend
  - ltrSmootherSTR.c, [397](#)
- fasp\_smoother\_dstr\_sor\_cf
  - ltrSmootherSTR.c, [397](#)
- fasp\_smoother\_dstr\_sor\_descend
  - ltrSmootherSTR.c, [398](#)
- fasp\_smoother\_dstr\_sor\_order
  - ltrSmootherSTR.c, [399](#)
- fasp\_solver\_amg
  - SolAMG.c, [527](#)
- fasp\_solver\_amli
  - PreMGRecurAMLI.c, [515](#)
- fasp\_solver\_dbic\_itsolver
  - SolBLC.c, [529](#)

fasp\_solver\_dblc\_krylov  
  SolBLC.c, [529](#)

fasp\_solver\_dblc\_krylov\_block\_3  
  SolBLC.c, [530](#)

fasp\_solver\_dblc\_krylov\_block\_4  
  SolBLC.c, [531](#)

fasp\_solver\_dblc\_krylov\_sweeping  
  SolBLC.c, [532](#)

fasp\_solver\_dblc\_pbcgs  
  KryPbcgs.c, [401](#)

fasp\_solver\_dblc\_pcg  
  KryPcg.c, [407](#)

fasp\_solver\_dblc\_pgmres  
  KryPgmres.c, [416](#)

fasp\_solver\_dblc\_pminres  
  KryPminres.c, [422](#)

fasp\_solver\_dblc\_pvbcgs  
  KryPvbcgs.c, [427](#)

fasp\_solver\_dblc\_pvfgmres  
  KryPvfgmres.c, [433](#)

fasp\_solver\_dblc\_pvgmres  
  KryPvgmres.c, [438](#)

fasp\_solver\_dblc\_spbcs  
  KrySPbcgs.c, [444](#)

fasp\_solver\_dblc\_spcg  
  KrySPcg.c, [449](#)

fasp\_solver\_dblc\_spgmres  
  KrySPgmres.c, [453](#)

fasp\_solver\_dblc\_spmminres  
  KrySPminres.c, [457](#)

fasp\_solver\_dblc\_spvgmres  
  KrySPvgmres.c, [460](#)

fasp\_solver\_dbsr\_itsolver  
  SolBSR.c, [534](#)

fasp\_solver\_dbsr\_krylov  
  SolBSR.c, [534](#)

fasp\_solver\_dbsr\_krylov\_amg  
  SolBSR.c, [535](#)

fasp\_solver\_dbsr\_krylov\_amg\_nk  
  SolBSR.c, [536](#)

fasp\_solver\_dbsr\_krylov\_diag  
  SolBSR.c, [537](#)

fasp\_solver\_dbsr\_krylov\_ilu  
  SolBSR.c, [537](#)

fasp\_solver\_dbsr\_krylov\_nk\_amg  
  SolBSR.c, [538](#)

fasp\_solver\_dbsr\_pbcgs  
  KryPbcgs.c, [401](#)

fasp\_solver\_dbsr\_pcg  
  KryPcg.c, [408](#)

fasp\_solver\_dbsr\_pgmres  
  KryPgmres.c, [417](#)

fasp\_solver\_dbsr\_pvbcgs  
  KryPvbcgs.c, [428](#)

fasp\_solver\_dbsr\_pvfgmres  
  KryPvfgmres.c, [434](#)

fasp\_solver\_dbsr\_pvgmres  
  KryPvgmres.c, [439](#)

fasp\_solver\_dbsr\_spbcs  
  KrySPbcgs.c, [444](#)

fasp\_solver\_dbsr\_spgmres  
  KrySPgmres.c, [454](#)

fasp\_solver\_dbsr\_spvgmres  
  KrySPvgmres.c, [461](#)

fasp\_solver\_dcsr\_itsolver  
  SolCSR.c, [540](#)

fasp\_solver\_dcsr\_krylov  
  SolCSR.c, [541](#)

fasp\_solver\_dcsr\_krylov\_amg  
  SolCSR.c, [542](#)

fasp\_solver\_dcsr\_krylov\_amg\_nk  
  SolCSR.c, [542](#)

fasp\_solver\_dcsr\_krylov\_diag  
  SolCSR.c, [543](#)

fasp\_solver\_dcsr\_krylov\_ilu  
  SolCSR.c, [544](#)

fasp\_solver\_dcsr\_krylov\_ilu\_M  
  SolCSR.c, [545](#)

fasp\_solver\_dcsr\_krylov\_schwarz  
  SolCSR.c, [545](#)

fasp\_solver\_dcsr\_pbcgs  
  KryPbcgs.c, [402](#)

fasp\_solver\_dcsr\_pcg  
  KryPcg.c, [409](#)

fasp\_solver\_dcsr\_pgcg  
  KryPgcg.c, [412](#)

fasp\_solver\_dcsr\_pgcr  
  KryPgcr.c, [415](#)

fasp\_solver\_dcsr\_pgmres  
  KryPgmres.c, [418](#)

fasp\_solver\_dcsr\_pminres  
  KryPminres.c, [423](#)

fasp\_solver\_dcsr\_pvbcgs  
  KryPvbcgs.c, [429](#)

fasp\_solver\_dcsr\_pvfgmres  
  KryPvfgmres.c, [435](#)

fasp\_solver\_dcsr\_pvgmres  
  KryPvgmres.c, [440](#)

fasp\_solver\_dcsr\_spbcs  
  KrySPbcgs.c, [445](#)

fasp\_solver\_dcsr\_spcg  
  KrySPcg.c, [449](#)

fasp\_solver\_dcsr\_spgmres  
  KrySPgmres.c, [455](#)

fasp\_solver\_dcsr\_spmminres  
  KrySPminres.c, [458](#)

fasp\_solver\_dcsr\_spvgmres  
  KrySPvgmres.c, [462](#)

fasp\_solver\_dstr\_itsolver  
  SolISTR.c, [559](#)

fasp\_solver\_dstr\_krylov  
  SolISTR.c, [560](#)

fasp\_solver\_dstr\_krylov\_blockgs  
  SolISTR.c, [561](#)

fasp\_solver\_dstr\_krylov\_diag  
  SolISTR.c, [561](#)

fasp\_solver\_dstr\_krylov\_ilu  
  SolISTR.c, [562](#)

fasp\_solver\_dstr\_pbcgs  
  KryPbcgs.c, [403](#)

fasp\_solver\_dstr\_pcg  
  KryPcg.c, [410](#)

fasp\_solver\_dstr\_pgmres  
  KryPgmres.c, [419](#)

fasp\_solver\_dstr\_pminres  
  KryPminres.c, [424](#)

fasp\_solver\_dstr\_pvbcs  
  KryPvbcs.c, [429](#)

fasp\_solver\_dstr\_pvgmres  
  KryPvgmres.c, [441](#)

fasp\_solver\_dstr\_spbcs  
  KrySPbcgs.c, [447](#)

fasp\_solver\_dstr\_spcg  
  KrySPcg.c, [450](#)

fasp\_solver\_dstr\_spgmres  
  KrySPgmres.c, [455](#)

fasp\_solver\_dstr\_spmminres  
  KrySPminres.c, [459](#)

fasp\_solver\_dstr\_spvgmres  
  KrySPvgmres.c, [463](#)

fasp\_solver\_famg  
  SolFAMG.c, [547](#)

fasp\_solver\_fmecycle  
  PreMGCycleFull.c, [512](#)

fasp\_solver\_itsolver  
  SolMatFree.c, [556](#)

fasp\_solver\_krylov  
  SolMatFree.c, [557](#)

fasp\_solver\_matfree\_init  
  SolMatFree.c, [558](#)

fasp\_solver\_mgcycle  
  PreMGCycle.c, [510](#)

fasp\_solver\_mgcycle\_bsr  
  PreMGCycle.c, [511](#)

fasp\_solver\_mgrecur  
  PreMGRecur.c, [513](#)

fasp\_solver\_mumps  
  XtrMumps.c, [568](#)

fasp\_solver\_mumps\_steps  
  XtrMumps.c, [569](#)

fasp\_solver\_nl\_amli  
  PreMGRecurAMLI.c, [516](#)

fasp\_solver\_nl\_amli\_bsr  
  PreMGRecurAMLI.c, [517](#)

fasp\_solver\_pardiso  
  XtrPardiso.c, [570](#)

fasp\_solver\_pbcgs  
  KryPbcgs.c, [404](#)

fasp\_solver\_pcg  
  KryPcg.c, [411](#)

fasp\_solver\_pgcg  
  KryPgcg.c, [413](#)

fasp\_solver\_pgmres  
  KryPgmres.c, [420](#)

fasp\_solver\_pminres  
  KryPminres.c, [425](#)

fasp\_solver\_pvbcs  
  KryPvbcs.c, [430](#)

fasp\_solver\_pvfgmres  
  KryPvfgmres.c, [436](#)

fasp\_solver\_pvgmres  
  KryPvgmres.c, [442](#)

fasp\_solver\_superlu  
  XtrSuperlu.c, [573](#)

fasp\_solver\_umfpack  
  XtrUmfpack.c, [574](#)

fasp\_sparse\_MIS  
  BlaSparseUtil.c, [277](#)

fasp\_sparse\_aat\_  
  BlaSparseUtil.c, [272](#)

fasp\_sparse\_abyb\_  
  BlaSparseUtil.c, [273](#)

fasp\_sparse\_abybms\_  
  BlaSparseUtil.c, [274](#)

fasp\_sparse\_aplbms\_  
  BlaSparseUtil.c, [275](#)

fasp\_sparse\_aplusb\_  
  BlaSparseUtil.c, [275](#)

fasp\_sparse\_iit\_  
  BlaSparseUtil.c, [276](#)

fasp\_sparse\_rapcmp\_  
  BlaSparseUtil.c, [277](#)

fasp\_sparse\_rapms\_  
  BlaSparseUtil.c, [278](#)

fasp\_sparse\_wta\_  
  BlaSparseUtil.c, [279](#)

fasp\_sparse\_wtams\_  
  BlaSparseUtil.c, [280](#)

fasp\_sparse\_ytx\_  
  BlaSparseUtil.c, [281](#)

fasp\_sparse\_ytxbig\_  
  BlaSparseUtil.c, [281](#)

fasp\_symbfactor  
  BlalLU.c, [151](#)

fasp\_topological\_sorting\_ilu  
  AuxSort.c, [118](#)

- BlalO.c, [186](#)
- BlalO.c, [187](#)
- SolWrapper.c, [565](#)
- SolWrapper.c, [566](#)
- G0PT
  - fasp\_const.h, [339](#)
- GE
  - fasp.h, [316](#)
- grid2d, [32](#)
  - e, [33](#)
  - edges, [33](#)
  - ediri, [33](#)
  - efather, [33](#)
  - fasp\_grid.h, [360](#)
  - p, [34](#)
  - pdiri, [34](#)
  - pfather, [34](#)
  - s, [34](#)
  - t, [34](#)
  - tfather, [35](#)
  - triangles, [35](#)
  - vertices, [35](#)
- GT
  - fasp.h, [317](#)
- iBLCmat, [35](#)
  - fasp\_block.h, [326](#)
- ICNTL
  - XtrMumps.c, [568](#)
- iCOOmat, [36](#)
  - fasp.h, [321](#)
- iCSRmat, [37](#)
  - fasp.h, [322](#)
- ILU\_MC\_OMP
  - fasp\_const.h, [339](#)
- ILU\_data, [38](#)
- ILU\_droptol
  - input\_param, [48](#)
- ILU\_ifil
  - input\_param, [48](#)
- ILU\_param, [40](#)
- ILU\_permtol
  - input\_param, [49](#)
- ILU\_relax
  - input\_param, [49](#)
- ILU\_type
  - input\_param, [49](#)
- ILUk
  - fasp\_const.h, [339](#)
- ILUt
  - fasp\_const.h, [340](#)
- ILUtp
  - fasp\_const.h, [340](#)
- IMAP
  - fasp.h, [322](#)
- INTERP\_DIR
  - fasp\_const.h, [340](#)
- INTERP\_ENG
  - fasp\_const.h, [340](#)
- INTERP\_EXT
  - fasp\_const.h, [341](#)
- INTERP\_STD
  - fasp\_const.h, [341](#)
- INT
  - fasp.h, [317](#)
- ISNAN
  - fasp.h, [317](#)
- ISPT
  - fasp\_const.h, [341](#)
- idenmat, [38](#)
  - fasp.h, [322](#)
- ilength
  - BlalO.c, [188](#)
- ilu\_solve\_omp
  - ltrSmootherBSR.c, [374](#)
- inifile
  - input\_param, [49](#)
- input\_param, [40](#)
  - AMG\_ILU\_levels, [44](#)
  - AMG\_Schwarz\_levels, [46](#)
  - AMG\_aggregation\_type, [42](#)
  - AMG\_aggressive\_level, [42](#)
  - AMG\_aggressive\_path, [42](#)
  - AMG\_amli\_degree, [42](#)
  - AMG\_coarse\_dof, [43](#)
  - AMG\_coarse\_scaling, [43](#)
  - AMG\_coarse\_solver, [43](#)
  - AMG\_coarsening\_type, [43](#)
  - AMG\_cycle\_type, [43](#)
  - AMG\_interpolation\_type, [44](#)
  - AMG\_levels, [44](#)
  - AMG\_max\_aggregation, [44](#)
  - AMG\_max\_row\_sum, [44](#)
  - AMG\_maxit, [45](#)
  - AMG\_nl\_amli\_krylov\_type, [45](#)
  - AMG\_pair\_number, [45](#)
  - AMG\_polynomial\_degree, [45](#)
  - AMG\_postsmooth\_iter, [45](#)
  - AMG\_presmooth\_iter, [46](#)
  - AMG\_quality\_bound, [46](#)
  - AMG\_relaxation, [46](#)
  - AMG\_smooth\_filter, [46](#)
  - AMG\_smooth\_order, [47](#)
  - AMG\_smoother, [47](#)



- AMG\_strong\_coupled, [47](#)
- AMG\_strong\_threshold, [47](#)
- AMG\_tentative\_smooth, [47](#)
- AMG\_tol, [48](#)
- AMG\_truncation\_threshold, [48](#)
- AMG\_type, [48](#)
- ILU\_droptol, [48](#)
- ILU\_lfil, [48](#)
- ILU\_permtol, [49](#)
- ILU\_relax, [49](#)
- ILU\_type, [49](#)
- inifile, [49](#)
- itsolver\_maxit, [49](#)
- itsolver\_tol, [50](#)
- output\_type, [50](#)
- precond\_type, [50](#)
- print\_level, [50](#)
- problem\_num, [50](#)
- restart, [51](#)
- Schwarz\_blk solver, [51](#)
- Schwarz\_maxlvl, [51](#)
- Schwarz\_mmsize, [51](#)
- Schwarz\_type, [51](#)
- solver\_type, [52](#)
- stop\_type, [52](#)
- workdir, [52](#)
- ltrSmootherBSR.c, [361](#)
  - fasp\_smoother\_dbsr\_gs, [362](#)
  - fasp\_smoother\_dbsr\_gs1, [363](#)
  - fasp\_smoother\_dbsr\_gs\_ascend, [364](#)
  - fasp\_smoother\_dbsr\_gs\_ascend1, [364](#)
  - fasp\_smoother\_dbsr\_gs\_descend, [365](#)
  - fasp\_smoother\_dbsr\_gs\_descend1, [366](#)
  - fasp\_smoother\_dbsr\_gs\_order1, [366](#)
  - fasp\_smoother\_dbsr\_gs\_order2, [367](#)
  - fasp\_smoother\_dbsr\_ilu, [368](#)
  - fasp\_smoother\_dbsr\_jacobi, [368](#)
  - fasp\_smoother\_dbsr\_jacobi1, [369](#)
  - fasp\_smoother\_dbsr\_jacobi\_setup, [369](#)
  - fasp\_smoother\_dbsr\_sor, [370](#)
  - fasp\_smoother\_dbsr\_sor1, [371](#)
  - fasp\_smoother\_dbsr\_sor\_ascend, [371](#)
  - fasp\_smoother\_dbsr\_sor\_descend, [372](#)
  - fasp\_smoother\_dbsr\_sor\_order, [373](#)
  - ilu\_solve\_omp, [374](#)
- ltrSmootherCSR.c, [374](#)
  - fasp\_smoother\_dcsr\_L1diag, [380](#)
  - fasp\_smoother\_dcsr\_gs, [375](#)
  - fasp\_smoother\_dcsr\_gs\_cf, [376](#)
  - fasp\_smoother\_dcsr\_gs\_rb3d, [376](#)
  - fasp\_smoother\_dcsr\_ilu, [377](#)
  - fasp\_smoother\_dcsr\_jacobi, [378](#)
  - fasp\_smoother\_dcsr\_kaczmarz, [379](#)
  - fasp\_smoother\_dcsr\_sgs, [380](#)
  - fasp\_smoother\_dcsr\_sor, [381](#)
  - fasp\_smoother\_dcsr\_sor\_cf, [382](#)
- ltrSmootherCSRcr.c, [383](#)
  - fasp\_smoother\_dcsr\_gscr, [383](#)
- ltrSmootherCSRpoly.c, [384](#)
  - fasp\_smoother\_dcsr\_poly, [385](#)
  - fasp\_smoother\_dcsr\_poly\_old, [385](#)
- ltrSmootherSTR.c, [386](#)
  - fasp\_generate\_diaginv\_block, [388](#)
  - fasp\_smoother\_dstr\_gs, [388](#)
  - fasp\_smoother\_dstr\_gs1, [389](#)
  - fasp\_smoother\_dstr\_gs\_ascend, [390](#)
  - fasp\_smoother\_dstr\_gs\_cf, [390](#)
  - fasp\_smoother\_dstr\_gs\_descend, [392](#)
  - fasp\_smoother\_dstr\_gs\_order, [393](#)
  - fasp\_smoother\_dstr\_jacobi, [393](#)
  - fasp\_smoother\_dstr\_jacobi1, [394](#)
  - fasp\_smoother\_dstr\_schwarz, [394](#)
  - fasp\_smoother\_dstr\_sor, [395](#)
  - fasp\_smoother\_dstr\_sor1, [396](#)
  - fasp\_smoother\_dstr\_sor\_ascend, [397](#)
  - fasp\_smoother\_dstr\_sor\_cf, [397](#)
  - fasp\_smoother\_dstr\_sor\_descend, [398](#)
  - fasp\_smoother\_dstr\_sor\_order, [399](#)
- itsolver\_maxit
  - input\_param, [49](#)
- itsolver\_param, [52](#)
  - itsolver\_type, [53](#)
  - maxit, [53](#)
  - precond\_type, [53](#)
  - print\_level, [53](#)
  - restart, [54](#)
  - stop\_type, [54](#)
  - tol, [54](#)
- itsolver\_tol
  - input\_param, [50](#)
- itsolver\_type
  - itsolver\_param, [53](#)
- ivector, [54](#)
  - fasp.h, [322](#)
- JA
  - dBSRmat, [27](#)
- KryPbcgs.c, [400](#)
  - fasp\_solver\_dblc\_pbcgs, [401](#)
  - fasp\_solver\_dbsr\_pbcgs, [401](#)
  - fasp\_solver\_dcsr\_pbcgs, [402](#)
  - fasp\_solver\_dstr\_pbcgs, [403](#)
  - fasp\_solver\_pbcgs, [404](#)
- KryPcg.c, [405](#)
  - fasp\_solver\_dblc\_pcg, [407](#)
  - fasp\_solver\_dbsr\_pcg, [408](#)
  - fasp\_solver\_dcsr\_pcg, [409](#)
  - fasp\_solver\_dstr\_pcg, [410](#)

- fasp\_solver\_pcg, 411
- KryPgcr.c, 412
  - fasp\_solver\_dcsr\_pgcr, 412
  - fasp\_solver\_pgcr, 413
- KryPgcr.c, 414
  - fasp\_solver\_dcsr\_pgcr, 415
- KryPgmres.c, 416
  - fasp\_solver\_dblc\_pgmres, 416
  - fasp\_solver\_dbsr\_pgmres, 417
  - fasp\_solver\_dcsr\_pgmres, 418
  - fasp\_solver\_dstr\_pgmres, 419
  - fasp\_solver\_pgmres, 420
- KryPminres.c, 422
  - fasp\_solver\_dblc\_pminres, 422
  - fasp\_solver\_dcsr\_pminres, 423
  - fasp\_solver\_dstr\_pminres, 424
  - fasp\_solver\_pminres, 425
- KryPvbcgs.c, 426
  - fasp\_solver\_dblc\_pvbcgs, 427
  - fasp\_solver\_dbsr\_pvbcgs, 428
  - fasp\_solver\_dcsr\_pvbcgs, 429
  - fasp\_solver\_dstr\_pvbcgs, 429
  - fasp\_solver\_pvbcgs, 430
- KryPvfgmres.c, 432
  - fasp\_solver\_dblc\_pvfgmres, 433
  - fasp\_solver\_dbsr\_pvfgmres, 434
  - fasp\_solver\_dcsr\_pvfgmres, 435
  - fasp\_solver\_pvfgmres, 436
- KryPvgmres.c, 437
  - fasp\_solver\_dblc\_pvgmres, 438
  - fasp\_solver\_dbsr\_pvgmres, 439
  - fasp\_solver\_dcsr\_pvgmres, 440
  - fasp\_solver\_dstr\_pvgmres, 441
  - fasp\_solver\_pvgmres, 442
- KrySPbcgs.c, 443
  - fasp\_solver\_dblc\_spbcgs, 444
  - fasp\_solver\_dbsr\_spbcgs, 444
  - fasp\_solver\_dcsr\_spbcgs, 445
  - fasp\_solver\_dstr\_spbcgs, 447
- KrySPcg.c, 448
  - fasp\_solver\_dblc\_spcg, 449
  - fasp\_solver\_dcsr\_spcg, 449
  - fasp\_solver\_dstr\_spcg, 450
- KrySPgmres.c, 452
  - fasp\_solver\_dblc\_spgmres, 453
  - fasp\_solver\_dbsr\_spgmres, 454
  - fasp\_solver\_dcsr\_spgmres, 455
  - fasp\_solver\_dstr\_spgmres, 455
- KrySPminres.c, 456
  - fasp\_solver\_dblc\_spmminres, 457
  - fasp\_solver\_dcsr\_spmminres, 458
  - fasp\_solver\_dstr\_spmminres, 459
- KrySPvgmres.c, 460
  - fasp\_solver\_dblc\_spvgmres, 460
  - fasp\_solver\_dbsr\_spvgmres, 461
  - fasp\_solver\_dcsr\_spvgmres, 462
  - fasp\_solver\_dstr\_spvgmres, 463
- LONGLONG
  - fasp.h, 318
- LONG
  - fasp.h, 318
- LU\_diag
  - precond\_block\_data, 62
- LE
  - fasp.h, 317
- local\_LU
  - precond\_sweeping\_data, 70
- local\_A
  - precond\_sweeping\_data, 70
- local\_index
  - precond\_sweeping\_data, 70
- LS
  - fasp.h, 318
- MAT\_BLC
  - fasp\_const.h, 342
- MAT\_BSR
  - fasp\_const.h, 342
- MAT\_CSR
  - fasp\_const.h, 342
- MAT\_CSR
  - fasp\_const.h, 342
- MAT\_FREE
  - fasp\_const.h, 343
- MAT\_STR
  - fasp\_const.h, 343
- MAT\_SymCSR
  - fasp\_const.h, 343
- MAT\_bBSR
  - fasp\_const.h, 341
- MAT\_bCSR
  - fasp\_const.h, 341
- MAT\_bSTR
  - fasp\_const.h, 342
- MAX\_AMG\_LVL
  - fasp\_const.h, 343
- MAX\_CRATE
  - fasp\_const.h, 343
- MAX\_REFINE\_LVL
  - fasp\_const.h, 344
- MAX\_RESTART
  - fasp\_const.h, 344
- MAX\_STAG
  - fasp\_const.h, 344
- MAXIMAP
  - fasp.h, 322
- MAX
  - fasp.h, 318

MIN\_CDOF  
    fasp\_const.h, 344  
MIN\_CRATE  
    fasp\_const.h, 344  
MIN  
    fasp.h, 319  
mallinfo, 55  
malloc\_chunk, 56  
malloc\_params, 56  
malloc\_segment, 56  
malloc\_state, 57  
malloc\_tree\_chunk, 58  
maxit  
    itsolver\_param, 53  
mgl  
    precond\_block\_data, 62  
Mumps\_data, 58  
mxv\_matfree, 59  
  
NEDMALLOC  
    fasp.h, 319  
NL\_AMLI\_CYCLE  
    fasp\_const.h, 345  
NO\_ORDER  
    fasp\_const.h, 345  
nedmallinfo, 59  
NumLayers  
    precond\_sweeping\_data, 70  
nx\_rb  
    fasp.h, 323  
ny\_rb  
    fasp.h, 323  
nz\_rb  
    fasp.h, 323  
  
OFF  
    fasp\_const.h, 345  
OPENMP\_HOLDS  
    fasp\_const.h, 346  
ON  
    fasp\_const.h, 345  
output\_type  
    input\_param, 50  
  
p  
    grid2d, 34  
PAIRWISE  
    fasp\_const.h, 346  
PREC\_AMG  
    fasp\_const.h, 346  
PREC\_DIAG  
    fasp\_const.h, 346  
PREC\_FMG  
    fasp\_const.h, 347  
PREC\_ILU  
    fasp\_const.h, 347  
PREC\_NULL  
    fasp\_const.h, 347  
PREC\_SCHWARZ  
    fasp\_const.h, 347  
PRINT\_ALL  
    fasp\_const.h, 347  
PRINT\_MIN  
    fasp\_const.h, 348  
PRINT\_MORE  
    fasp\_const.h, 348  
PRINT\_MOST  
    fasp\_const.h, 348  
PRINT\_NONE  
    fasp\_const.h, 348  
PRINT\_SOME  
    fasp\_const.h, 348  
PUT\_INT  
    fasp.h, 319  
PUT\_REAL  
    fasp.h, 319  
Pardiso\_data, 60  
pcgrid2d  
    fasp\_grid.h, 360  
pdiri  
    grid2d, 34  
pfather  
    grid2d, 34  
pgrid2d  
    fasp\_grid.h, 361  
PreAMGCoarsenCR.c, 464  
    fasp\_amg\_coarsening\_cr, 465  
PreAMGCoarsenRS.c, 466  
    fasp\_amg\_coarsening\_rs, 466  
PreAMGInterp.c, 467  
    fasp\_amg\_interp, 468  
    fasp\_amg\_interp\_trunc, 469  
PreAMGInterpEM.c, 469  
    fasp\_amg\_interp\_em, 470  
PreAMGSetupCR.c, 471  
    fasp\_amg\_setup\_cr, 471  
PreAMGSetupRS.c, 472  
    fasp\_amg\_setup\_rs, 473  
PreAMGSetupSA.c, 473  
    fasp\_amg\_setup\_sa, 474  
PreAMGSetupSABSR.c, 475  
    fasp\_amg\_setup\_sa\_bsr, 475  
PreAMGSetupUA.c, 476  
    fasp\_amg\_setup\_ua, 477  
PreAMGSetupUABSR.c, 477  
    fasp\_amg\_setup\_ua\_bsr, 478  
PreBLC.c, 479  
    fasp\_precond\_block\_SGS\_3, 483  
    fasp\_precond\_block\_SGS\_3\_amg, 484

- fasp\_precond\_block\_diag\_3, 480
  - fasp\_precond\_block\_diag\_3\_amg, 480
  - fasp\_precond\_block\_diag\_4, 481
  - fasp\_precond\_block\_lower\_3, 481
  - fasp\_precond\_block\_lower\_3\_amg, 482
  - fasp\_precond\_block\_lower\_4, 482
  - fasp\_precond\_block\_upper\_3, 484
  - fasp\_precond\_block\_upper\_3\_amg, 485
  - fasp\_precond\_sweeping, 485
- PreBSR.c, 486
  - fasp\_precond\_dbsr\_amg, 487
  - fasp\_precond\_dbsr\_amg\_nk, 487
  - fasp\_precond\_dbsr\_diag, 488
  - fasp\_precond\_dbsr\_diag\_nc2, 489
  - fasp\_precond\_dbsr\_diag\_nc3, 489
  - fasp\_precond\_dbsr\_diag\_nc5, 490
  - fasp\_precond\_dbsr\_diag\_nc7, 491
  - fasp\_precond\_dbsr\_ilu, 492
  - fasp\_precond\_dbsr\_ilu\_ls\_omp, 492
  - fasp\_precond\_dbsr\_ilu\_mc\_omp, 493
  - fasp\_precond\_dbsr\_nl\_amli, 494
- PreCSR.c, 494
  - fasp\_precond\_amg, 495
  - fasp\_precond\_amg\_nk, 496
  - fasp\_precond\_amli, 496
  - fasp\_precond\_diag, 497
  - fasp\_precond\_famg, 498
  - fasp\_precond\_free, 498
  - fasp\_precond\_ilu, 499
  - fasp\_precond\_ilu\_backward, 499
  - fasp\_precond\_ilu\_forward, 501
  - fasp\_precond\_nl\_amli, 501
  - fasp\_precond\_schwarz, 502
  - fasp\_precond\_setup, 503
- PreDataInit.c, 503
  - fasp\_amg\_data\_bsr\_create, 504
  - fasp\_amg\_data\_bsr\_free, 505
  - fasp\_amg\_data\_create, 505
  - fasp\_amg\_data\_free, 506
  - fasp\_ilu\_data\_create, 506
  - fasp\_ilu\_data\_free, 507
  - fasp\_ilu\_data\_null, 507
  - fasp\_precond\_data\_null, 508
  - fasp\_precond\_null, 508
  - fasp\_schwarz\_data\_free, 509
- PreMGCycle.c, 510
  - fasp\_solver\_mgcycle, 510
  - fasp\_solver\_mgcycle\_bsr, 511
- PreMGCycleFull.c, 511
  - fasp\_solver\_fmgcycle, 512
- PreMGRecur.c, 513
  - fasp\_solver\_mgrecur, 513
- PreMGRecurAMLI.c, 514
  - fasp\_amg\_amli\_coef, 515
  - fasp\_solver\_amli, 515
  - fasp\_solver\_nl\_amli, 516
  - fasp\_solver\_nl\_amli\_bsr, 517
- PreMGSolve.c, 518
  - fasp\_amg\_solve, 519
  - fasp\_amg\_solve\_amli, 519
  - fasp\_amg\_solve\_nl\_amli, 520
  - fasp\_famg\_solve, 521
- PreSTR.c, 521
  - fasp\_precond\_dstr\_blockgs, 522
  - fasp\_precond\_dstr\_diag, 523
  - fasp\_precond\_dstr\_ilu0, 523
  - fasp\_precond\_dstr\_ilu0\_backward, 524
  - fasp\_precond\_dstr\_ilu0\_forward, 524
  - fasp\_precond\_dstr\_ilu1, 525
  - fasp\_precond\_dstr\_ilu1\_backward, 526
  - fasp\_precond\_dstr\_ilu1\_forward, 526
- precond, 60
- precond\_block\_data, 61
  - A\_diag, 61
  - Ablc, 61
  - amgparam, 61
  - LU\_diag, 62
  - mgl, 62
  - r, 62
- precond\_data, 62
- precond\_data\_bsr, 64
- precond\_data\_str, 66
- precond\_diagbsr, 67
- precond\_diagstr, 68
- precond\_sweeping\_data, 69
  - A, 69
  - Ai, 69
  - local\_LU, 70
  - local\_A, 70
  - local\_index, 70
  - NumLayers, 70
  - r, 70
  - w, 71
- precond\_type
  - input\_param, 50
  - itsolver\_param, 53
- print\_amgcomplexity
  - AuxMessage.c, 96
- print\_amgcomplexity\_bsr
  - AuxMessage.c, 97
- print\_cputime
  - AuxMessage.c, 97
- print\_itinfo
  - AuxMessage.c, 98
- print\_level
  - input\_param, 50
  - itsolver\_param, 53
- print\_message

- AuxMessage.c, [99](#)
- problem\_num
  - input\_param, [50](#)
- r
  - precond\_block\_data, [62](#)
  - precond\_sweeping\_data, [70](#)
- REAL
  - fasp.h, [320](#)
- RS\_C1
  - fasp.h, [320](#)
- restart
  - input\_param, [51](#)
  - itsolver\_param, [54](#)
- s
  - grid2d, [34](#)
- SA\_AMG
  - fasp\_const.h, [349](#)
- SCHWARZ\_BACKWARD
  - fasp\_const.h, [349](#)
- SCHWARZ\_FORWARD
  - fasp\_const.h, [349](#)
- SCHWARZ\_SYMMETRIC
  - fasp\_const.h, [349](#)
- SHORT
  - fasp.h, [320](#)
- SMALLREAL2
  - fasp\_const.h, [350](#)
- SMALLREAL
  - fasp\_const.h, [349](#)
- SMOOTHER\_BLKOil
  - fasp\_const.h, [350](#)
- SMOOTHER\_CG
  - fasp\_const.h, [350](#)
- SMOOTHER\_GSOR
  - fasp\_const.h, [350](#)
- SMOOTHER\_GS
  - fasp\_const.h, [350](#)
- SMOOTHER\_JACOBI
  - fasp\_const.h, [351](#)
- SMOOTHER\_L1DIAG
  - fasp\_const.h, [351](#)
- SMOOTHER\_POLY
  - fasp\_const.h, [351](#)
- SMOOTHER\_SGSOR
  - fasp\_const.h, [351](#)
- SMOOTHER\_SGS
  - fasp\_const.h, [351](#)
- SMOOTHER\_SOR
  - fasp\_const.h, [352](#)
- SMOOTHER\_SPETEN
  - fasp\_const.h, [352](#)
- SMOOTHER\_SSOR
  - fasp\_const.h, [352](#)
- SOLVER\_AMG
  - fasp\_const.h, [352](#)
- SOLVER\_BiCGstab
  - fasp\_const.h, [352](#)
- SOLVER\_CG
  - fasp\_const.h, [353](#)
- SOLVER\_DEFAULT
  - fasp\_const.h, [353](#)
- SOLVER\_FMG
  - fasp\_const.h, [353](#)
- SOLVER\_GCG
  - fasp\_const.h, [353](#)
- SOLVER\_GCR
  - fasp\_const.h, [353](#)
- SOLVER\_GMRES
  - fasp\_const.h, [354](#)
- SOLVER\_MUMPS
  - fasp\_const.h, [354](#)
- SOLVER\_MinRes
  - fasp\_const.h, [354](#)
- SOLVER\_PARDISO
  - fasp\_const.h, [354](#)
- SOLVER\_SBiCGstab
  - fasp\_const.h, [354](#)
- SOLVER\_SCG
  - fasp\_const.h, [355](#)
- SOLVER\_SGCG
  - fasp\_const.h, [355](#)
- SOLVER\_SGMRES
  - fasp\_const.h, [355](#)
- SOLVER\_SMinRes
  - fasp\_const.h, [355](#)
- SOLVER\_SUPERLU
  - fasp\_const.h, [355](#)
- SOLVER\_SVFGMRES
  - fasp\_const.h, [356](#)
- SOLVER\_SVGMRES
  - fasp\_const.h, [356](#)
- SOLVER\_UMFPACK
  - fasp\_const.h, [356](#)
- SOLVER\_VBiCGstab
  - fasp\_const.h, [356](#)
- SOLVER\_VFGMRES
  - fasp\_const.h, [356](#)
- SOLVER\_VGMRES
  - fasp\_const.h, [357](#)
- STAG\_RATIO
  - fasp\_const.h, [357](#)
- STOP\_MOD\_REL\_RES
  - fasp\_const.h, [357](#)
- STOP\_REL\_PRECRES
  - fasp\_const.h, [357](#)
- STOP\_REL\_RES
  - fasp\_const.h, [357](#)

- SWAP
  - BlaSmallMatInv.c, 216
- Schwarz\_blk solver
  - input\_param, 51
- Schwarz\_data, 71
- Schwarz\_maxlvl
  - input\_param, 51
- Schwarz\_mmsize
  - input\_param, 51
- Schwarz\_param, 72
- Schwarz\_type
  - input\_param, 51
- SolAMG.c, 527
  - fasp\_solver\_amg, 527
- SolBLC.c, 528
  - fasp\_solver\_dblc\_itsolver, 529
  - fasp\_solver\_dblc\_krylov, 529
  - fasp\_solver\_dblc\_krylov\_block\_3, 530
  - fasp\_solver\_dblc\_krylov\_block\_4, 531
  - fasp\_solver\_dblc\_krylov\_sweeping, 532
- SolBSR.c, 533
  - fasp\_solver\_dbsr\_itsolver, 534
  - fasp\_solver\_dbsr\_krylov, 534
  - fasp\_solver\_dbsr\_krylov\_amg, 535
  - fasp\_solver\_dbsr\_krylov\_amg\_nk, 536
  - fasp\_solver\_dbsr\_krylov\_diag, 537
  - fasp\_solver\_dbsr\_krylov\_ilu, 537
  - fasp\_solver\_dbsr\_krylov\_nk\_amg, 538
- SolCSR.c, 539
  - fasp\_solver\_dcsr\_itsolver, 540
  - fasp\_solver\_dcsr\_krylov, 541
  - fasp\_solver\_dcsr\_krylov\_amg, 542
  - fasp\_solver\_dcsr\_krylov\_amg\_nk, 542
  - fasp\_solver\_dcsr\_krylov\_diag, 543
  - fasp\_solver\_dcsr\_krylov\_ilu, 544
  - fasp\_solver\_dcsr\_krylov\_ilu\_M, 545
  - fasp\_solver\_dcsr\_krylov\_schwarz, 545
- SolFAMG.c, 546
  - fasp\_solver\_famg, 547
- SolGMGPoisson.c, 548
  - fasp\_poisson\_fgmg1d, 549
  - fasp\_poisson\_fgmg2d, 549
  - fasp\_poisson\_fgmg3d, 550
  - fasp\_poisson\_gmg1d, 551
  - fasp\_poisson\_gmg2d, 551
  - fasp\_poisson\_gmg3d, 552
  - fasp\_poisson\_gmgcg1d, 553
  - fasp\_poisson\_gmgcg2d, 554
  - fasp\_poisson\_gmgcg3d, 555
- SolMatFree.c, 556
  - fasp\_solver\_itsolver, 556
  - fasp\_solver\_krylov, 557
  - fasp\_solver\_matfree\_init, 558
- SolSTR.c, 559
  - fasp\_solver\_dstr\_itsolver, 559
  - fasp\_solver\_dstr\_krylov, 560
  - fasp\_solver\_dstr\_krylov\_blockgs, 561
  - fasp\_solver\_dstr\_krylov\_diag, 561
  - fasp\_solver\_dstr\_krylov\_ilu, 562
- SolWrapper.c, 563
  - fasp\_fwrapper\_amg, 564
  - fasp\_fwrapper\_krylov\_amg, 564
  - fasp\_wrapper\_dbsr\_krylov\_amg, 565
  - fasp\_wrapper\_dcoo\_dbsr\_krylov\_amg, 566
- solver\_type
  - input\_param, 52
- stop\_type
  - input\_param, 52
  - itsolver\_param, 54
- t
  - grid2d, 34
- THDs\_AMG\_GS
  - AuxThreads.c, 121
- THDs\_CPR\_gGS
  - AuxThreads.c, 121
- THDs\_CPR\_IGS
  - AuxThreads.c, 121
- TRUE
  - fasp\_const.h, 358
- tfather
  - grid2d, 35
- tol
  - itsolver\_param, 54
- total\_alloc\_count
  - AuxMemory.c, 94
  - fasp.h, 323
- total\_alloc\_mem
  - AuxMemory.c, 95
  - fasp.h, 323
- triangles
  - grid2d, 35
- UA\_AMG
  - fasp\_const.h, 358
- UNPT
  - fasp\_const.h, 358
- USERDEFINED
  - fasp\_const.h, 358
- V\_CYCLE
  - fasp\_const.h, 359
- VMB
  - fasp\_const.h, 359
- val
  - dBSRmat, 27
- vertices
  - grid2d, 35

## w

- precond\_sweeping\_data, [71](#)

## W\_CYCLE

- fasp\_const.h, [359](#)

## workdir

- input\_param, [52](#)

XtrMumps.c, [567](#)

- fasp\_solver\_mumps, [568](#)

- fasp\_solver\_mumps\_steps, [569](#)

- ICNTL, [568](#)

XtrPardiso.c, [570](#)

- fasp\_solver\_pardiso, [570](#)

XtrSamg.c, [571](#)

- dCSRmat2SAMGInput, [571](#)

- dvector2SAMGInput, [572](#)

XtrSuperlu.c, [572](#)

- fasp\_solver\_superlu, [573](#)

XtrUmfpack.c, [574](#)

- fasp\_solver\_umfpack, [574](#)