

## Fast Auxiliary Space Preconditioning

1.8.2 Oct/18/2015

Generated by Doxygen 1.8.10

Mon Oct 19 2015 08:38:57



# 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>Todo List</b>	<b>11</b>
<b>7</b>	<b>Data Structure Index</b>	<b>13</b>
7.1	Data Structures . . . . .	13
<b>8</b>	<b>File Index</b>	<b>17</b>
8.1	File List . . . . .	17
<b>9</b>	<b>Data Structure Documentation</b>	<b>23</b>
9.1	AMG_data Struct Reference . . . . .	23
9.1.1	Detailed Description . . . . .	24
9.2	AMG_data_bsr Struct Reference . . . . .	24
9.2.1	Detailed Description . . . . .	25
9.3	AMG_param Struct Reference . . . . .	26
9.3.1	Detailed Description . . . . .	28
9.4	block_BSR Struct Reference . . . . .	28
9.4.1	Detailed Description . . . . .	28
9.5	block_dCSRmat Struct Reference . . . . .	28
9.5.1	Detailed Description . . . . .	29
9.6	block_dvector Struct Reference . . . . .	29
9.6.1	Detailed Description . . . . .	29
9.7	block_iCSRmat Struct Reference . . . . .	29

9.7.1 Detailed Description . . . . .	30
9.8 block_ivector Struct Reference . . . . .	30
9.8.1 Detailed Description . . . . .	30
9.9 block_Reservoir Struct Reference . . . . .	31
9.9.1 Detailed Description . . . . .	31
9.10 dBSRmat Struct Reference . . . . .	31
9.10.1 Detailed Description . . . . .	32
9.10.2 Field Documentation . . . . .	32
9.10.2.1 JA . . . . .	32
9.10.2.2 val . . . . .	32
9.11 dCOOmat Struct Reference . . . . .	32
9.11.1 Detailed Description . . . . .	33
9.12 dCSRLmat Struct Reference . . . . .	33
9.12.1 Detailed Description . . . . .	34
9.13 dCSRmat Struct Reference . . . . .	34
9.13.1 Detailed Description . . . . .	34
9.14 ddenmat Struct Reference . . . . .	34
9.14.1 Detailed Description . . . . .	35
9.15 dSTRmat Struct Reference . . . . .	35
9.15.1 Detailed Description . . . . .	36
9.16 dvector Struct Reference . . . . .	36
9.16.1 Detailed Description . . . . .	36
9.17 grid2d Struct Reference . . . . .	36
9.17.1 Detailed Description . . . . .	37
9.17.2 Field Documentation . . . . .	37
9.17.2.1 e . . . . .	37
9.17.2.2 edges . . . . .	37
9.17.2.3 ediri . . . . .	37
9.17.2.4 efather . . . . .	37
9.17.2.5 p . . . . .	37
9.17.2.6 pdiri . . . . .	38
9.17.2.7 pfather . . . . .	38
9.17.2.8 s . . . . .	38
9.17.2.9 t . . . . .	38
9.17.2.10 tfather . . . . .	38
9.17.2.11 triangles . . . . .	38
9.17.2.12 vertices . . . . .	38

9.18 iCOOmat Struct Reference . . . . .	38
9.18.1 Detailed Description . . . . .	39
9.19 iCSRmat Struct Reference . . . . .	39
9.19.1 Detailed Description . . . . .	40
9.20 idenmat Struct Reference . . . . .	40
9.20.1 Detailed Description . . . . .	40
9.21 ILU_data Struct Reference . . . . .	40
9.21.1 Detailed Description . . . . .	41
9.22 ILU_param Struct Reference . . . . .	41
9.22.1 Detailed Description . . . . .	42
9.23 input_param Struct Reference . . . . .	42
9.23.1 Detailed Description . . . . .	43
9.23.2 Field Documentation . . . . .	43
9.23.2.1 AMG_aggregation_type . . . . .	43
9.23.2.2 AMG_aggressive_level . . . . .	43
9.23.2.3 AMG_aggressive_path . . . . .	43
9.23.2.4 AMG_amli_degree . . . . .	43
9.23.2.5 AMG_coarse_dof . . . . .	44
9.23.2.6 AMG_coarse_scaling . . . . .	44
9.23.2.7 AMG_coarse_solver . . . . .	44
9.23.2.8 AMG_coarsening_type . . . . .	44
9.23.2.9 AMG_cycle_type . . . . .	44
9.23.2.10 AMG_ILU_levels . . . . .	44
9.23.2.11 AMG_interpolation_type . . . . .	44
9.23.2.12 AMG_levels . . . . .	44
9.23.2.13 AMG_max_aggregation . . . . .	44
9.23.2.14 AMG_max_row_sum . . . . .	45
9.23.2.15 AMG_maxit . . . . .	45
9.23.2.16 AMG_nl_amli_krylov_type . . . . .	45
9.23.2.17 AMG_pair_number . . . . .	45
9.23.2.18 AMG_polynomial_degree . . . . .	45
9.23.2.19 AMG_postsmooth_iter . . . . .	45
9.23.2.20 AMG_presmooth_iter . . . . .	45
9.23.2.21 AMG_quality_bound . . . . .	45
9.23.2.22 AMG_relaxation . . . . .	45
9.23.2.23 AMG_Schwarz_levels . . . . .	46
9.23.2.24 AMG_smooth_filter . . . . .	46

9.23.2.25 AMG_smooth_order . . . . .	46
9.23.2.26 AMG_smoother . . . . .	46
9.23.2.27 AMG_strong_coupled . . . . .	46
9.23.2.28 AMG_strong_threshold . . . . .	46
9.23.2.29 AMG_tentative_smooth . . . . .	46
9.23.2.30 AMG_tol . . . . .	46
9.23.2.31 AMG_truncation_threshold . . . . .	46
9.23.2.32 AMG_type . . . . .	47
9.23.2.33 ILU_droptol . . . . .	47
9.23.2.34 ILU_lfil . . . . .	47
9.23.2.35 ILU_permtol . . . . .	47
9.23.2.36 ILU_relax . . . . .	47
9.23.2.37 ILU_type . . . . .	47
9.23.2.38 inifile . . . . .	47
9.23.2.39 itsolver_maxit . . . . .	47
9.23.2.40 itsolver_tol . . . . .	47
9.23.2.41 output_type . . . . .	48
9.23.2.42 precondition_type . . . . .	48
9.23.2.43 print_level . . . . .	48
9.23.2.44 problem_num . . . . .	48
9.23.2.45 restart . . . . .	48
9.23.2.46 Schwarz_blkssolver . . . . .	48
9.23.2.47 Schwarz_maxlvl . . . . .	48
9.23.2.48 Schwarz_mmsize . . . . .	48
9.23.2.49 Schwarz_type . . . . .	48
9.23.2.50 solver_type . . . . .	49
9.23.2.51 stop_type . . . . .	49
9.23.2.52 workdir . . . . .	49
9.24 itsolver_param Struct Reference . . . . .	49
9.24.1 Detailed Description . . . . .	49
9.24.2 Field Documentation . . . . .	49
9.24.2.1 itsolver_type . . . . .	49
9.24.2.2 maxit . . . . .	50
9.24.2.3 precondition_type . . . . .	50
9.24.2.4 print_level . . . . .	50
9.24.2.5 restart . . . . .	50
9.24.2.6 stop_type . . . . .	50

9.24.2.7 tol	50
9.25 ivector Struct Reference	50
9.25.1 Detailed Description	51
9.26 Link Struct Reference	51
9.26.1 Detailed Description	51
9.27 linked_list Struct Reference	51
9.27.1 Detailed Description	52
9.28 mallinfo Struct Reference	52
9.28.1 Detailed Description	52
9.29 malloc_chunk Struct Reference	52
9.29.1 Detailed Description	53
9.30 malloc_params Struct Reference	53
9.30.1 Detailed Description	53
9.31 malloc_segment Struct Reference	53
9.31.1 Detailed Description	53
9.32 malloc_state Struct Reference	53
9.32.1 Detailed Description	54
9.33 malloc_tree_chunk Struct Reference	54
9.33.1 Detailed Description	54
9.34 Mumps_data Struct Reference	55
9.34.1 Detailed Description	55
9.35 mxv_matfree Struct Reference	55
9.35.1 Detailed Description	55
9.36 nedmallinfo Struct Reference	55
9.36.1 Detailed Description	56
9.37 precondition Struct Reference	56
9.37.1 Detailed Description	56
9.38 precondition_block_data Struct Reference	56
9.38.1 Detailed Description	57
9.38.2 Field Documentation	57
9.38.2.1 A_diag	57
9.38.2.2 Abcsr	57
9.38.2.3 amgparam	57
9.38.2.4 LU_diag	57
9.38.2.5 mgl	57
9.38.2.6 r	58
9.39 precondition_block_reservoir_data Struct Reference	58

9.39.1 Detailed Description . . . . .	59
9.39.2 Field Documentation . . . . .	59
9.39.2.1 diag . . . . .	59
9.39.2.2 diaginv . . . . .	60
9.39.2.3 diaginvS . . . . .	60
9.39.2.4 order . . . . .	60
9.39.2.5 perf_idx . . . . .	60
9.39.2.6 pivot . . . . .	60
9.39.2.7 pivotS . . . . .	60
9.39.2.8 PP . . . . .	60
9.39.2.9 r . . . . .	60
9.39.2.10 RR . . . . .	60
9.39.2.11 scaled . . . . .	61
9.39.2.12 SS . . . . .	61
9.39.2.13 w . . . . .	61
9.39.2.14 WW . . . . .	61
9.40 precondition_data Struct Reference . . . . .	61
9.40.1 Detailed Description . . . . .	62
9.41 precondition_data_bsr Struct Reference . . . . .	63
9.41.1 Detailed Description . . . . .	64
9.42 precondition_data_str Struct Reference . . . . .	64
9.42.1 Detailed Description . . . . .	66
9.43 precondition_diagbsr Struct Reference . . . . .	66
9.43.1 Detailed Description . . . . .	66
9.44 precondition_diagstr Struct Reference . . . . .	66
9.44.1 Detailed Description . . . . .	67
9.45 precondition_FASP_blkcoil_data Struct Reference . . . . .	67
9.45.1 Detailed Description . . . . .	68
9.45.2 Field Documentation . . . . .	69
9.45.2.1 A . . . . .	69
9.45.2.2 diaginv . . . . .	69
9.45.2.3 diaginv_noscale . . . . .	69
9.45.2.4 diaginv_S . . . . .	69
9.45.2.5 maxit . . . . .	69
9.45.2.6 mgl_data . . . . .	69
9.45.2.7 neigh . . . . .	69
9.45.2.8 order . . . . .	69



9.45.2.9 perf_idx . . . . .	70
9.45.2.10 perf_neigh . . . . .	70
9.45.2.11 pivot . . . . .	70
9.45.2.12 pivot_S . . . . .	70
9.45.2.13 PP . . . . .	70
9.45.2.14 r . . . . .	70
9.45.2.15 restart . . . . .	70
9.45.2.16 RR . . . . .	70
9.45.2.17 scaled . . . . .	70
9.45.2.18 SS . . . . .	71
9.45.2.19 tol . . . . .	71
9.45.2.20 w . . . . .	71
9.45.2.21 WW . . . . .	71
9.46 preconditioning_data Struct Reference . . . . .	71
9.46.1 Detailed Description . . . . .	72
9.46.2 Field Documentation . . . . .	72
9.46.2.1 A . . . . .	72
9.46.2.2 Ai . . . . .	72
9.46.2.3 local_A . . . . .	72
9.46.2.4 local_index . . . . .	72
9.46.2.5 local_LU . . . . .	72
9.46.2.6 NumLayers . . . . .	73
9.46.2.7 r . . . . .	73
9.46.2.8 w . . . . .	73
9.47 Schwarz_data Struct Reference . . . . .	73
9.47.1 Detailed Description . . . . .	74
9.48 Schwarz_param Struct Reference . . . . .	74
9.48.1 Detailed Description . . . . .	75
<b>10 File Documentation</b> . . . . .	<b>77</b>
10.1 amg.c File Reference . . . . .	77
10.1.1 Detailed Description . . . . .	77
10.1.2 Function Documentation . . . . .	77
10.1.2.1 fasp_solver_amg(dCSRmat *A, dvector *b, dvector *x, AMG_param *param) . . . . .	77
10.2 amg_setup_cr.c File Reference . . . . .	78
10.2.1 Detailed Description . . . . .	78
10.2.2 Function Documentation . . . . .	78

10.2.2.1	<code>fasp_amg_setup_cr(AMG_data *mgl, AMG_param *param)</code>	78
10.3	<code>amg_setup_rs.c</code> File Reference	79
10.3.1	Detailed Description	79
10.3.2	Function Documentation	79
10.3.2.1	<code>fasp_amg_setup_rs(AMG_data *mgl, AMG_param *param)</code>	79
10.4	<code>amg_setup_sa.c</code> File Reference	80
10.4.1	Detailed Description	80
10.4.2	Function Documentation	81
10.4.2.1	<code>fasp_amg_setup_sa(AMG_data *mgl, AMG_param *param)</code>	81
10.4.2.2	<code>fasp_amg_setup_sa_bsr(AMG_data_bsr *mgl, AMG_param *param)</code>	81
10.5	<code>amg_setup_ua.c</code> File Reference	82
10.5.1	Detailed Description	82
10.5.2	Function Documentation	82
10.5.2.1	<code>fasp_amg_setup_ua(AMG_data *mgl, AMG_param *param)</code>	82
10.5.2.2	<code>fasp_amg_setup_ua_bsr(AMG_data_bsr *mgl, AMG_param *param)</code>	83
10.6	<code>amg_solve.c</code> File Reference	84
10.6.1	Detailed Description	84
10.6.2	Function Documentation	85
10.6.2.1	<code>fasp_amg_solve(AMG_data *mgl, AMG_param *param)</code>	85
10.6.2.2	<code>fasp_amg_solve_amli(AMG_data *mgl, AMG_param *param)</code>	86
10.6.2.3	<code>fasp_amg_solve_nl_amli(AMG_data *mgl, AMG_param *param)</code>	86
10.6.2.4	<code>fasp_famg_solve(AMG_data *mgl, AMG_param *param)</code>	87
10.7	<code>amlirecur.c</code> File Reference	87
10.7.1	Detailed Description	88
10.7.2	Function Documentation	88
10.7.2.1	<code>fasp_amg_amli_coef(const REAL lambda_max, const REAL lambda_min, const INT degree, REAL *coef)</code>	88
10.7.2.2	<code>fasp_solver_amli(AMG_data *mgl, AMG_param *param, INT level)</code>	88
10.7.2.3	<code>fasp_solver_nl_amli(AMG_data *mgl, AMG_param *param, INT level, INT num_levels)</code>	89
10.7.2.4	<code>fasp_solver_nl_amli_bsr(AMG_data_bsr *mgl, AMG_param *param, INT level, INT num_levels)</code>	89
10.8	<code>array.c</code> File Reference	90
10.8.1	Detailed Description	91
10.8.2	Function Documentation	91
10.8.2.1	<code>fasp_array_cp(const INT n, REAL *x, REAL *y)</code>	91
10.8.2.2	<code>fasp_array_cp_nc3(REAL *x, REAL *y)</code>	91
10.8.2.3	<code>fasp_array_cp_nc5(REAL *x, REAL *y)</code>	91

10.8.2.4	<code>fasp_array_cp_nc7(REAL *x, REAL *y)</code>	92
10.8.2.5	<code>fasp_array_null(REAL *x)</code>	92
10.8.2.6	<code>fasp_array_set(const INT n, REAL *x, const REAL val)</code>	93
10.8.2.7	<code>fasp_iarray_cp(const INT n, INT *x, INT *y)</code>	93
10.8.2.8	<code>fasp_iarray_set(const INT n, INT *x, const INT val)</code>	93
10.9	<code>blas_array.c</code> File Reference	94
10.9.1	Detailed Description	94
10.9.2	Function Documentation	95
10.9.2.1	<code>fasp_blas_array_ax(const INT n, const REAL a, REAL *x)</code>	95
10.9.2.2	<code>fasp_blas_array_axpby(const INT n, const REAL a, REAL *x, const REAL b, REAL *y)</code>	96
10.9.2.3	<code>fasp_blas_array_axpy(const INT n, const REAL a, REAL *x, REAL *y)</code>	96
10.9.2.4	<code>fasp_blas_array_axpyz(const INT n, const REAL a, REAL *x, REAL *y, REAL *z)</code>	97
10.9.2.5	<code>fasp_blas_array_dotprod(const INT n, const REAL *x, const REAL *y)</code>	97
10.9.2.6	<code>fasp_blas_array_norm1(const INT n, const REAL *x)</code>	98
10.9.2.7	<code>fasp_blas_array_norm2(const INT n, const REAL *x)</code>	98
10.9.2.8	<code>fasp_blas_array_norminf(const INT n, const REAL *x)</code>	99
10.10	<code>blas_bcsr.c</code> File Reference	99
10.10.1	Detailed Description	100
10.10.2	Function Documentation	100
10.10.2.1	<code>fasp_blas_bdbsr_aApy(const REAL alpha, block_BSR *A, REAL *x, REAL *y)</code>	100
10.10.2.2	<code>fasp_blas_bdbsr_mxv(block_BSR *A, REAL *x, REAL *y)</code>	100
10.10.2.3	<code>fasp_blas_bdcsr_aApy(const REAL alpha, block_dCSRmat *A, REAL *x, REAL *y)</code>	101
10.10.2.4	<code>fasp_blas_bdcsr_mxv(block_dCSRmat *A, REAL *x, REAL *y)</code>	101
10.11	<code>blas_bsr.c</code> File Reference	101
10.11.1	Detailed Description	102
10.11.2	Function Documentation	102
10.11.2.1	<code>fasp_blas_dbsr_aApy(const REAL alpha, dBSRmat *A, REAL *x, const REAL beta, REAL *y)</code>	102
10.11.2.2	<code>fasp_blas_dbsr_aApy(const REAL alpha, dBSRmat *A, REAL *x, REAL *y)</code>	103
10.11.2.3	<code>fasp_blas_dbsr_aApy_agg(const REAL alpha, dBSRmat *A, REAL *x, REAL *y)</code>	103
10.11.2.4	<code>fasp_blas_dbsr_axm(dBSRmat *A, const REAL alpha)</code>	104
10.11.2.5	<code>fasp_blas_dbsr_mxm(dBSRmat *A, dBSRmat *B, dBSRmat *C)</code>	105
10.11.2.6	<code>fasp_blas_dbsr_mxv(dBSRmat *A, REAL *x, REAL *y)</code>	105
10.11.2.7	<code>fasp_blas_dbsr_mxv_agg(dBSRmat *A, REAL *x, REAL *y)</code>	106
10.11.2.8	<code>fasp_blas_dbsr_rap(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	106
10.11.2.9	<code>fasp_blas_dbsr_rap1(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	107
10.11.2.10	<code>fasp_blas_dbsr_rap_agg(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	107

10.12	blas_csr.c File Reference	108
10.12.1	Detailed Description	108
10.12.2	Function Documentation	109
10.12.2.1	fasp_blas_dcsr_aApy(const REAL alpha, dCSRmat *A, REAL *x, REAL *y)	109
10.12.2.2	fasp_blas_dcsr_aApy_agg(const REAL alpha, dCSRmat *A, REAL *x, REAL *y)	109
10.12.2.3	fasp_blas_dcsr_add(dCSRmat *A, const REAL alpha, dCSRmat *B, const REAL beta, dCSRmat *C)	110
10.12.2.4	fasp_blas_dcsr_axm(dCSRmat *A, const REAL alpha)	110
10.12.2.5	fasp_blas_dcsr_bandwith(dCSRmat *A, INT *bndwith)	111
10.12.2.6	fasp_blas_dcsr_mxm(dCSRmat *A, dCSRmat *B, dCSRmat *C)	112
10.12.2.7	fasp_blas_dcsr_mxv(dCSRmat *A, REAL *x, REAL *y)	112
10.12.2.8	fasp_blas_dcsr_mxv_agg(dCSRmat *A, REAL *x, REAL *y)	113
10.12.2.9	fasp_blas_dcsr_ptap(dCSRmat *Pt, dCSRmat *A, dCSRmat *P, dCSRmat *Ac)	113
10.12.2.10	fasp_blas_dcsr_rap(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *RAP)	114
10.12.2.11	fasp_blas_dcsr_rap4(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *B, INT *icor_ysk)	115
10.12.2.12	fasp_blas_dcsr_rap_agg(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *RAP)	115
10.12.2.13	fasp_blas_dcsr_rap_agg1(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *B)	116
10.12.2.14	fasp_blas_dcsr_vmv(dCSRmat *A, REAL *x, REAL *y)	116
10.13	blas_csrl.c File Reference	117
10.13.1	Detailed Description	117
10.13.2	Function Documentation	117
10.13.2.1	fasp_blas_dcsrl_mxv(dCSRLmat *A, REAL *x, REAL *y)	117
10.14	blas_smat.c File Reference	118
10.14.1	Detailed Description	120
10.14.2	Function Documentation	120
10.14.2.1	fasp_blas_array_axpy_nc2(const REAL a, REAL *x, REAL *y)	120
10.14.2.2	fasp_blas_array_axpy_nc3(const REAL a, REAL *x, REAL *y)	120
10.14.2.3	fasp_blas_array_axpy_nc5(const REAL a, REAL *x, REAL *y)	120
10.14.2.4	fasp_blas_array_axpy_nc7(const REAL a, REAL *x, REAL *y)	121
10.14.2.5	fasp_blas_array_axpyz_nc2(const REAL a, REAL *x, REAL *y, REAL *z)	121
10.14.2.6	fasp_blas_array_axpyz_nc3(const REAL a, REAL *x, REAL *y, REAL *z)	122
10.14.2.7	fasp_blas_array_axpyz_nc5(const REAL a, REAL *x, REAL *y, REAL *z)	122
10.14.2.8	fasp_blas_array_axpyz_nc7(const REAL a, REAL *x, REAL *y, REAL *z)	123
10.14.2.9	fasp_blas_smat_aApyby(const REAL alpha, REAL *A, REAL *x, const REAL beta, REAL *y, const INT n)	124
10.14.2.10	fasp_blas_smat_add(REAL *a, REAL *b, const INT n, const REAL alpha, const REAL beta, REAL *c)	124

10.14.2.1	<code>fasp_blas_smat_axm</code> (REAL *a, const INT n, const REAL alpha)	125
10.14.2.2	<code>fasp_blas_smat_mul</code> (REAL *a, REAL *b, REAL *c, const INT n)	125
10.14.2.3	<code>fasp_blas_smat_mul_nc2</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.4	<code>fasp_blas_smat_mul_nc3</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.5	<code>fasp_blas_smat_mul_nc5</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.6	<code>fasp_blas_smat_mul_nc7</code> (REAL *a, REAL *b, REAL *c)	127
10.14.2.7	<code>fasp_blas_smat_m xv</code> (REAL *a, REAL *b, REAL *c, const INT n)	127
10.14.2.8	<code>fasp_blas_smat_m xv_nc2</code> (REAL *a, REAL *b, REAL *c)	128
10.14.2.9	<code>fasp_blas_smat_m xv_nc3</code> (REAL *a, REAL *b, REAL *c)	129
10.14.2.20	<code>fasp_blas_smat_m xv_nc5</code> (REAL *a, REAL *b, REAL *c)	129
10.14.2.21	<code>fasp_blas_smat_m xv_nc7</code> (REAL *a, REAL *b, REAL *c)	130
10.14.2.22	<code>fasp_blas_smat_ymAx</code> (REAL *A, REAL *x, REAL *y, const INT n)	131
10.14.2.23	<code>fasp_blas_smat_ymAx_nc2</code> (REAL *A, REAL *x, REAL *y)	131
10.14.2.24	<code>fasp_blas_smat_ymAx_nc3</code> (REAL *A, REAL *x, REAL *y)	132
10.14.2.25	<code>fasp_blas_smat_ymAx_nc5</code> (REAL *A, REAL *x, REAL *y)	132
10.14.2.26	<code>fasp_blas_smat_ymAx_nc7</code> (REAL *A, REAL *x, REAL *y)	132
10.14.2.27	<code>fasp_blas_smat_ymAx_ns</code> (REAL *A, REAL *x, REAL *y, const INT n)	133
10.14.2.28	<code>fasp_blas_smat_ymAx_ns2</code> (REAL *A, REAL *x, REAL *y)	133
10.14.2.29	<code>fasp_blas_smat_ymAx_ns3</code> (REAL *A, REAL *x, REAL *y)	134
10.14.2.30	<code>fasp_blas_smat_ymAx_ns5</code> (REAL *A, REAL *x, REAL *y)	134
10.14.2.31	<code>fasp_blas_smat_ymAx_ns7</code> (REAL *A, REAL *x, REAL *y)	135
10.14.2.32	<code>fasp_blas_smat_ypAx</code> (REAL *A, REAL *x, REAL *y, const INT n)	135
10.14.2.33	<code>fasp_blas_smat_ypAx_nc2</code> (REAL *A, REAL *x, REAL *y)	136
10.14.2.34	<code>fasp_blas_smat_ypAx_nc3</code> (REAL *A, REAL *x, REAL *y)	136
10.14.2.35	<code>fasp_blas_smat_ypAx_nc5</code> (REAL *A, REAL *x, REAL *y)	137
10.14.2.36	<code>fasp_blas_smat_ypAx_nc7</code> (REAL *A, REAL *x, REAL *y)	138
10.15	<code>blas_str.c</code> File Reference	138
10.15.1	Detailed Description	139
10.15.2	Function Documentation	139
10.15.2.1	<code>fasp_blas_dstr_aApy</code> (const REAL alpha, dSTRmat *A, REAL *x, REAL *y)	139
10.15.2.2	<code>fasp_blas_dstr_m xv</code> (dSTRmat *A, REAL *x, REAL *y)	139
10.15.2.3	<code>fasp_dstr_diagscale</code> (dSTRmat *A, dSTRmat *B)	139
10.16	<code>blas_vec.c</code> File Reference	140
10.16.1	Detailed Description	140
10.16.2	Function Documentation	141
10.16.2.1	<code>fasp_blas_dvec_axpy</code> (const REAL a, dvector *x, dvector *y)	141
10.16.2.2	<code>fasp_blas_dvec_axpyz</code> (const REAL a, dvector *x, dvector *y, dvector *z)	141

10.16.2.3 fasp_blas_dvec_dotprod(dvector *x, dvector *y)	141
10.16.2.4 fasp_blas_dvec_norm1(dvector *x)	142
10.16.2.5 fasp_blas_dvec_norm2(dvector *x)	142
10.16.2.6 fasp_blas_dvec_norminf(dvector *x)	143
10.16.2.7 fasp_blas_dvec_relerr(dvector *x, dvector *y)	143
10.17checkmat.c File Reference	144
10.17.1 Detailed Description	145
10.17.2 Function Documentation	145
10.17.2.1 fasp_check_dCSRmat(dCSRmat *A)	145
10.17.2.2 fasp_check_diagdom(dCSRmat *A)	145
10.17.2.3 fasp_check_diagpos(dCSRmat *A)	146
10.17.2.4 fasp_check_diagzero(dCSRmat *A)	147
10.17.2.5 fasp_check_iCSRmat(iCSRmat *A)	147
10.17.2.6 fasp_check_symm(dCSRmat *A)	148
10.18coarsening_cr.c File Reference	149
10.18.1 Detailed Description	149
10.18.2 Function Documentation	149
10.18.2.1 fasp_amg_coarsening_cr(const INT i_0, const INT i_n, dCSRmat *A, ivector *vertices, AMG_param *param)	149
10.19coarsening_rs.c File Reference	150
10.19.1 Detailed Description	150
10.19.2 Function Documentation	151
10.19.2.1 fasp_amg_coarsening_rs(dCSRmat *A, ivector *vertices, dCSRmat *P, iCSRmat *S, AMG_param *param)	151
10.20convert.c File Reference	152
10.20.1 Detailed Description	153
10.20.2 Function Documentation	153
10.20.2.1 endian_convert_int(const INT inum, const INT ilength, const INT endianflag)	153
10.20.2.2 endian_convert_real(const REAL rnum, const INT vlength, const INT endianflag)	153
10.20.2.3 fasp_aux_bbyteToldouble(unsigned char bytes[])	154
10.20.2.4 fasp_aux_change_endian4(unsigned long x)	155
10.20.2.5 fasp_aux_change_endian8(double x)	155
10.21doxygen.h File Reference	156
10.21.1 Detailed Description	156
10.22eigen.c File Reference	156
10.22.1 Detailed Description	156
10.22.2 Function Documentation	156

10.22.2.1 fasp_dcsr_eig(dCSRmat *A, const REAL tol, const INT maxit)	156
10.23 famg.c File Reference	157
10.23.1 Detailed Description	157
10.23.2 Function Documentation	157
10.23.2.1 fasp_solver_famg(dCSRmat *A, dvector *b, dvector *x, AMG_param *param)	157
10.24 fasp.h File Reference	158
10.24.1 Detailed Description	161
10.24.2 Macro Definition Documentation	161
10.24.2.1 __FASP_HEADER__	161
10.24.2.2 ABS	161
10.24.2.3 DIAGONAL_PREF	161
10.24.2.4 DLMALLOC	161
10.24.2.5 FASP_GSRB	161
10.24.2.6 FASP_USE_ILU	161
10.24.2.7 FASP_VERSION	162
10.24.2.8 GE	162
10.24.2.9 GT	162
10.24.2.10 INT	162
10.24.2.11 ISNAN	162
10.24.2.12 LE	162
10.24.2.13 LONG	162
10.24.2.14 LONGLONG	162
10.24.2.15 LS	163
10.24.2.16 MAX	163
10.24.2.17 MIN	163
10.24.2.18 NEDMALLOC	163
10.24.2.19 PUT_INT	163
10.24.2.20 PUT_REAL	163
10.24.2.21 REAL	163
10.24.2.22 RS_C1	163
10.24.2.23 SHORT	164
10.24.3 Typedef Documentation	164
10.24.3.1 dCOOmat	164
10.24.3.2 dCSRLmat	164
10.24.3.3 dCSRmat	164
10.24.3.4 ddenmat	164
10.24.3.5 dSTRmat	164

10.24.3.6	<a href="#">dvector</a>	164
10.24.3.7	<a href="#">grid2d</a>	164
10.24.3.8	<a href="#">iCOOmat</a>	164
10.24.3.9	<a href="#">iCSRmat</a>	165
10.24.3.10	<a href="#">idenmat</a>	165
10.24.3.11	<a href="#">ivector</a>	165
10.24.3.12	<a href="#">linkList</a>	165
10.24.3.13	<a href="#">listElement</a>	165
10.24.3.14	<a href="#">pcgrid2d</a>	165
10.24.3.15	<a href="#">pgrid2d</a>	165
10.24.4	<a href="#">Variable Documentation</a>	165
10.24.4.1	<a href="#">count</a>	165
10.24.4.2	<a href="#">IMAP</a>	165
10.24.4.3	<a href="#">MAXIMAP</a>	165
10.24.4.4	<a href="#">nx_rb</a>	166
10.24.4.5	<a href="#">ny_rb</a>	166
10.24.4.6	<a href="#">nz_rb</a>	166
10.24.4.7	<a href="#">total_alloc_count</a>	166
10.24.4.8	<a href="#">total_alloc_mem</a>	166
10.25	<a href="#">fasp_block.h File Reference</a>	166
10.25.1	<a href="#">Detailed Description</a>	167
10.25.2	<a href="#">Macro Definition Documentation</a>	168
10.25.2.1	<a href="#">__FASPBLOCK_HEADER__</a>	168
10.25.2.2	<a href="#">SMOOTHER_BLKOIL</a>	168
10.25.2.3	<a href="#">SMOOTHER_SPETEN</a>	168
10.25.3	<a href="#">Typedef Documentation</a>	168
10.25.3.1	<a href="#">block_BSR</a>	168
10.25.3.2	<a href="#">block_dCSRmat</a>	168
10.25.3.3	<a href="#">block_dvector</a>	168
10.25.3.4	<a href="#">block_iCSRmat</a>	168
10.25.3.5	<a href="#">block_ivector</a>	168
10.25.3.6	<a href="#">block_Reservoir</a>	168
10.25.3.7	<a href="#">dBSRmat</a>	169
10.25.3.8	<a href="#">precond_block_reservoir_data</a>	169
10.26	<a href="#">fasp_const.h File Reference</a>	169
10.26.1	<a href="#">Detailed Description</a>	172
10.26.2	<a href="#">Macro Definition Documentation</a>	172



10.26.2.1	AMLI_CYCLE	172
10.26.2.2	ASCEND	172
10.26.2.3	BIGREAL	173
10.26.2.4	CF_ORDER	173
10.26.2.5	CGPT	173
10.26.2.6	CLASSIC_AMG	173
10.26.2.7	COARSE_AC	173
10.26.2.8	COARSE_CR	173
10.26.2.9	COARSE_MIS	173
10.26.2.10	COARSE_RS	173
10.26.2.11	COARSE_RSP	174
10.26.2.12	CPFIRST	174
10.26.2.13	DESCEND	174
10.26.2.14	ERROR_ALLOC_MEM	174
10.26.2.15	ERROR_AMG_COARSE_TYPE	174
10.26.2.16	ERROR_AMG_COARSEING	174
10.26.2.17	ERROR_AMG_INTERP_TYPE	174
10.26.2.18	ERROR_AMG_SMOOTH_TYPE	174
10.26.2.19	ERROR_DATA_STRUCTURE	174
10.26.2.20	ERROR_DATA_ZERODIAG	175
10.26.2.21	ERROR_DUMMY_VAR	175
10.26.2.22	ERROR_INPUT_PAR	175
10.26.2.23	ERROR_LIC_TYPE	175
10.26.2.24	ERROR_MAT_SIZE	175
10.26.2.25	ERROR_MISC	175
10.26.2.26	ERROR_NUM_BLOCKS	175
10.26.2.27	ERROR_OPEN_FILE	175
10.26.2.28	ERROR_QUAD_DIM	175
10.26.2.29	ERROR_QUAD_TYPE	176
10.26.2.30	ERROR_REGRESS	176
10.26.2.31	ERROR_SOLVER_EXIT	176
10.26.2.32	ERROR_SOLVER_ILUSETUP	176
10.26.2.33	ERROR_SOLVER_MAXIT	176
10.26.2.34	ERROR_SOLVER_MISC	176
10.26.2.35	ERROR_SOLVER_PRECTYPE	176
10.26.2.36	ERROR_SOLVER_SOLSTAG	176
10.26.2.37	ERROR_SOLVER_STAG	176

10.26.2.38	ERROR_SOLVER_TOLSMALL	177
10.26.2.39	ERROR_SOLVER_TYPE	177
10.26.2.40	ERROR_UNKNOWN	177
10.26.2.41	ERROR_WRONG_FILE	177
10.26.2.42	FALSE	177
10.26.2.43	FASP_SUCCESS	177
10.26.2.44	FGPT	177
10.26.2.45	FPFIRST	177
10.26.2.46	G0PT	178
10.26.2.47	LUk	178
10.26.2.48	LUt	178
10.26.2.49	LUtp	178
10.26.2.50	INTERP_DIR	178
10.26.2.51	INTERP_ENG	178
10.26.2.52	INTERP_STD	178
10.26.2.53	ISPT	178
10.26.2.54	MAT_bBSR	179
10.26.2.55	MAT_bCSR	179
10.26.2.56	MAT_BSR	179
10.26.2.57	MAT_CSR	179
10.26.2.58	MAT_CSRL	179
10.26.2.59	MAT_FREE	179
10.26.2.60	MAT_STR	179
10.26.2.61	MAT_SymCSR	179
10.26.2.62	MAX_AMG_LVL	180
10.26.2.63	MAX_CRATE	180
10.26.2.64	MAX_REFINE_LVL	180
10.26.2.65	MAX_RESTART	180
10.26.2.66	MAX_STAG	180
10.26.2.67	MIN_CDOF	180
10.26.2.68	MIN_CRATE	180
10.26.2.69	NL_AMLI_CYCLE	180
10.26.2.70	NO_ORDER	180
10.26.2.71	OFF	181
10.26.2.72	ON	181
10.26.2.73	OPENMP_HOLDS	181
10.26.2.74	PAIRWISE	181

10.26.2.75	PREC_AMG	181
10.26.2.76	PREC_DIAG	181
10.26.2.77	PREC_FMG	181
10.26.2.78	PREC_ILU	181
10.26.2.79	PREC_NULL	182
10.26.2.80	PREC_SCHWARZ	182
10.26.2.81	PRINT_ALL	182
10.26.2.82	PRINT_MIN	182
10.26.2.83	PRINT_MORE	182
10.26.2.84	PRINT_MOST	182
10.26.2.85	PRINT_NONE	182
10.26.2.86	PRINT_SOME	182
10.26.2.87	SA_AMG	183
10.26.2.88	SCHWARZ_BACKWARD	183
10.26.2.89	SCHWARZ_FORWARD	183
10.26.2.90	SCHWARZ_SYMMETRIC	183
10.26.2.91	SMALLREAL	183
10.26.2.92	SMALLREAL2	183
10.26.2.93	SMOOTHER_CG	183
10.26.2.94	SMOOTHER_GS	183
10.26.2.95	SMOOTHER_GSOR	184
10.26.2.96	SMOOTHER_JACOBI	184
10.26.2.97	SMOOTHER_L1DIAG	184
10.26.2.98	SMOOTHER_POLY	184
10.26.2.99	SMOOTHER_SGS	184
10.26.2.100	SMOOTHER_SGSOR	184
10.26.2.101	SMOOTHER_SOR	184
10.26.2.102	SMOOTHER_SSOR	184
10.26.2.103	SOLVER_AMG	185
10.26.2.104	SOLVER_BiCGstab	185
10.26.2.105	SOLVER_CG	185
10.26.2.106	SOLVER_DEFAULT	185
10.26.2.107	SOLVER_FMG	185
10.26.2.108	SOLVER_GCG	185
10.26.2.109	SOLVER_GCR	185
10.26.2.110	SOLVER_GMRES	185
10.26.2.111	SOLVER_MinRes	186

10.26.2.11	<del>S</del> SOLVER_MUMPS	186
10.26.2.11	<del>S</del> SOLVER_SBiCGstab	186
10.26.2.11	<del>S</del> SOLVER_SCG	186
10.26.2.11	<del>S</del> SOLVER_SGCG	186
10.26.2.11	<del>S</del> SOLVER_SGMRES	186
10.26.2.11	<del>S</del> SOLVER_SMinRes	186
10.26.2.11	<del>S</del> SOLVER_SUPERLU	186
10.26.2.11	<del>S</del> SOLVER_SVFGMRES	186
10.26.2.12	<del>S</del> SOLVER_SVGMRES	187
10.26.2.12	<del>S</del> SOLVER_UMFPACK	187
10.26.2.12	<del>S</del> SOLVER_VFGMRES	187
10.26.2.12	<del>S</del> SOLVER_VGMRES	187
10.26.2.12	<del>S</del> TAG_RATIO	187
10.26.2.12	<del>S</del> TOP_MOD_REL_RES	187
10.26.2.12	<del>S</del> TOP_REL_PRECRES	187
10.26.2.12	<del>S</del> TOP_REL_RES	187
10.26.2.12	<del>S</del> TRUE	188
10.26.2.12	<del>S</del> A_AMG	188
10.26.2.13	<del>S</del> UNPT	188
10.26.2.13	<del>S</del> USERDEFINED	188
10.26.2.13	<del>S</del> _CYCLE	188
10.26.2.13	<del>S</del> MB	188
10.26.2.13	<del>S</del> _CYCLE	188
10.27	fmgcycle.c File Reference	188
10.27.1	Detailed Description	189
10.27.2	Function Documentation	189
10.27.2.1	fasp_solver_fmgcycle(AMG_data *mgl, AMG_param *param)	189
10.28	formats.c File Reference	189
10.28.1	Detailed Description	190
10.28.2	Function Documentation	190
10.28.2.1	fasp_format_bdcsr_dcsr(block_dCSRmat *Ab)	190
10.28.2.2	fasp_format_dbsr_dcoo(dBSRmat *B)	190
10.28.2.3	fasp_format_dbsr_dcsr(dBSRmat *B)	191
10.28.2.4	fasp_format_dcoo_dcsr(dCOOmat *A, dCSRmat *B)	191
10.28.2.5	fasp_format_dcsr_dbsr(dCSRmat *A, const INT nb)	192
10.28.2.6	fasp_format_dcsr_dcoo(dCSRmat *A, dCOOmat *B)	192
10.28.2.7	fasp_format_dcsr_dcsr(dCSRmat *A)	193

10.28.2.8 fasp_format_dstr_dbsr(dSTRmat *B)	193
10.28.2.9 fasp_format_dstr_dcsr(dSTRmat *A, dCSRmat *B)	194
10.29givens.c File Reference	194
10.29.1 Detailed Description	195
10.29.2 Function Documentation	195
10.29.2.1 fasp_aux_givens(const REAL beta, dCSRmat *H, dvector *y, REAL *tmp)	195
10.30gmg_poisson.c File Reference	195
10.30.1 Detailed Description	196
10.30.2 Function Documentation	196
10.30.2.1 fasp_poisson_fgmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	196
10.30.2.2 fasp_poisson_fgmg_2D(REAL *u, REAL *b, const INT nx, const INT ny, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	196
10.30.2.3 fasp_poisson_fgmg_3D(REAL *u, REAL *b, const INT nx, const INT ny, const INT nz, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	197
10.30.2.4 fasp_poisson_gmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	197
10.30.2.5 fasp_poisson_gmg_2D(REAL *u, REAL *b, const INT nx, const INT ny, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	198
10.30.2.6 fasp_poisson_gmg_3D(REAL *u, REAL *b, const INT nx, const INT ny, const INT nz, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	198
10.30.2.7 fasp_poisson_pcg_gmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	199
10.30.2.8 fasp_poisson_pcg_gmg_2D(REAL *u, REAL *b, const INT nx, const INT ny, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	200
10.30.2.9 fasp_poisson_pcg_gmg_3D(REAL *u, REAL *b, const INT nx, const INT ny, const INT nz, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	201
10.31graphics.c File Reference	202
10.31.1 Detailed Description	202
10.31.2 Function Documentation	202
10.31.2.1 fasp_dbsr_plot(const dBSRmat *A, const char *fname)	202
10.31.2.2 fasp_dbsr_subplot(const dBSRmat *A, const char *filename, INT size)	203
10.31.2.3 fasp_dcsr_plot(const dCSRmat *A, const char *fname)	203
10.31.2.4 fasp_dcsr_subplot(const dCSRmat *A, const char *filename, INT size)	204
10.31.2.5 fasp_grid2d_plot(pgrid2d pg, INT level)	204
10.32ilu_setup_bsr.c File Reference	205
10.32.1 Detailed Description	205
10.32.2 Function Documentation	205
10.32.2.1 fasp_ilu_dbsr_setup(dBSRmat *A, ILU_data *iludata, ILU_param *iluparam)	205
10.33ilu_setup_csr.c File Reference	206

10.33.1 Detailed Description	206
10.33.2 Function Documentation	206
10.33.2.1 fasp_ilu_dcsr_setup(dCSRmat *A, ILU_data *iludata, ILU_param *iluparam)	206
10.34ilu_setup_str.c File Reference	207
10.34.1 Detailed Description	207
10.34.2 Function Documentation	207
10.34.2.1 fasp_ilu_dstr_setup0(dSTRmat *A, dSTRmat *LU)	207
10.34.2.2 fasp_ilu_dstr_setup1(dSTRmat *A, dSTRmat *LU)	208
10.35init.c File Reference	208
10.35.1 Detailed Description	209
10.35.2 Function Documentation	209
10.35.2.1 fasp_amg_data_bsr_create(SHORT max_levels)	209
10.35.2.2 fasp_amg_data_bsr_free(AMG_data_bsr *mgl)	210
10.35.2.3 fasp_amg_data_create(SHORT max_levels)	210
10.35.2.4 fasp_amg_data_free(AMG_data *mgl, AMG_param *param)	210
10.35.2.5 fasp_ilu_data_alloc(const INT iwk, const INT nwork, ILU_data *iludata)	211
10.35.2.6 fasp_ilu_data_free(ILU_data *ILUdata)	211
10.35.2.7 fasp_ilu_data_null(ILU_data *ILUdata)	211
10.35.2.8 fasp_precond_data_null(precond_data *pcdata)	212
10.35.2.9 fasp_precond_null(precond *pcdata)	212
10.35.2.10 fasp_Schwarz_data_free(Schwarz_data *Schwarz)	212
10.36input.c File Reference	213
10.36.1 Detailed Description	213
10.36.2 Function Documentation	213
10.36.2.1 fasp_param_check(input_param *inparam)	213
10.36.2.2 fasp_param_input(const char *filenm, input_param *inparam)	214
10.37interface_mumps.c File Reference	214
10.37.1 Detailed Description	215
10.37.2 Macro Definition Documentation	215
10.37.2.1 ICNTL	215
10.37.3 Function Documentation	215
10.37.3.1 fasp_solver_mumps(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)	215
10.37.3.2 fasp_solver_mumps_steps(dCSRmat *ptrA, dvector *b, dvector *u, Mumps_data *mumps)	215
10.38interface_samg.c File Reference	216
10.38.1 Detailed Description	216
10.38.2 Function Documentation	216

10.38.2.1 dCSRmat2SAMGInput(dCSRmat *A, char *filefrm, char *fileamg) . . . . .	216
10.38.2.2 dvector2SAMGInput(dvector *vec, char *filename) . . . . .	217
10.39 interface_superlu.c File Reference . . . . .	217
10.39.1 Detailed Description . . . . .	217
10.39.2 Function Documentation . . . . .	217
10.39.2.1 fasp_solver_superlu(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl) . . . . .	217
10.40 interface_umfpack.c File Reference . . . . .	218
10.40.1 Detailed Description . . . . .	218
10.40.2 Function Documentation . . . . .	218
10.40.2.1 fasp_solver_umfpack(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl) . . . . .	218
10.41 interpolation.c File Reference . . . . .	219
10.41.1 Detailed Description . . . . .	219
10.41.2 Function Documentation . . . . .	219
10.41.2.1 fasp_amg_interp(dCSRmat *A, ivecator *vertices, dCSRmat *P, iCSRmat *S, AMG_param *param) . . . . .	219
10.41.2.2 fasp_amg_interp1(dCSRmat *A, ivecator *vertices, dCSRmat *P, AMG_param *param, iCSRmat *S, INT *icor_ysk) . . . . .	220
10.41.2.3 fasp_amg_interp_trunc(dCSRmat *P, AMG_param *param) . . . . .	220
10.42 interpolation_em.c File Reference . . . . .	221
10.42.1 Detailed Description . . . . .	221
10.42.2 Function Documentation . . . . .	221
10.42.2.1 fasp_amg_interp_em(dCSRmat *A, ivecator *vertices, dCSRmat *P, AMG_param *param) . . . . .	221
10.43 io.c File Reference . . . . .	222
10.43.1 Detailed Description . . . . .	224
10.43.2 Function Documentation . . . . .	224
10.43.2.1 fasp_dbsr_print(dBSRmat *A) . . . . .	224
10.43.2.2 fasp_dbsr_read(const char *filename, dBSRmat *A) . . . . .	224
10.43.2.3 fasp_dbsr_write(const char *filename, dBSRmat *A) . . . . .	225
10.43.2.4 fasp_dbsr_write_coo(const char *filename, const dBSRmat *A) . . . . .	225
10.43.2.5 fasp_dcoo1_read(const char *filename, dCOOmat *A) . . . . .	226
10.43.2.6 fasp_dcoo_print(dCOOmat *A) . . . . .	226
10.43.2.7 fasp_dcoo_read(const char *filename, dCSRmat *A) . . . . .	227
10.43.2.8 fasp_dcoo_shift_read(const char *filename, dCSRmat *A) . . . . .	227
10.43.2.9 fasp_dcoo_write(const char *filename, dCSRmat *A) . . . . .	228
10.43.2.10 fasp_dcsr_print(dCSRmat *A) . . . . .	228
10.43.2.11 fasp_dcsr_read(const char *filename, dCSRmat *A) . . . . .	229
10.43.2.12 fasp_dcsr_write_coo(const char *filename, const dCSRmat *A) . . . . .	229

10.43.2.13	<code>asp_dcsrvec1_read(const char *filename, dCSRmat *A, dvector *b)</code>	229
10.43.2.14	<code>asp_dcsrvec1_write(const char *filename, dCSRmat *A, dvector *b)</code>	230
10.43.2.15	<code>asp_dcsrvec2_read(const char *filename, const char *filerhs, dCSRmat *A, dvector *b)</code>	231
10.43.2.16	<code>asp_dcsrvec2_write(const char *filename, const char *filerhs, dCSRmat *A, dvector *b)</code>	231
10.43.2.17	<code>asp_dmtx_read(const char *filename, dCSRmat *A)</code>	232
10.43.2.18	<code>asp_dmtxsym_read(const char *filename, dCSRmat *A)</code>	233
10.43.2.19	<code>asp_dstr_print(dSTRmat *A)</code>	234
10.43.2.20	<code>asp_dstr_read(const char *filename, dSTRmat *A)</code>	234
10.43.2.21	<code>asp_dstr_write(const char *filename, dSTRmat *A)</code>	235
10.43.2.22	<code>asp_dvec_print(INT n, dvector *u)</code>	235
10.43.2.23	<code>asp_dvec_read(const char *filename, dvector *b)</code>	236
10.43.2.24	<code>asp_dvec_write(const char *filename, dvector *vec)</code>	236
10.43.2.25	<code>asp_dvecind_read(const char *filename, dvector *b)</code>	237
10.43.2.26	<code>asp_dvecind_write(const char *filename, dvector *vec)</code>	237
10.43.2.27	<code>asp_hb_read(const char *input_file, dCSRmat *A, dvector *b)</code>	237
10.43.2.28	<code>asp_ivec_print(INT n, ivec *u)</code>	238
10.43.2.29	<code>asp_ivec_read(const char *filename, ivec *b)</code>	238
10.43.2.30	<code>asp_ivec_write(const char *filename, ivec *vec)</code>	239
10.43.2.31	<code>asp_ivecind_read(const char *filename, ivec *b)</code>	239
10.43.2.32	<code>asp_matrix_read(const char *filename, void *A)</code>	240
10.43.2.33	<code>asp_matrix_read_bin(const char *filename, void *A)</code>	240
10.43.2.34	<code>asp_matrix_write(const char *filename, void *A, INT flag)</code>	241
10.43.2.35	<code>asp_vector_read(const char *filerhs, void *b)</code>	241
10.43.2.36	<code>asp_vector_write(const char *filerhs, void *b, INT flag)</code>	242
10.43.3	Variable Documentation	243
10.43.3.1	<code>dlength</code>	243
10.43.3.2	<code>ilength</code>	243
10.44	<code>itsolver_bcsr.c</code> File Reference	243
10.44.1	Detailed Description	244
10.44.2	Function Documentation	244
10.44.2.1	<code>asp_solver_bdcsr_itsolver(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	244
10.44.2.2	<code>asp_solver_bdcsr_krylov(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	244
10.44.2.3	<code>asp_solver_bdcsr_krylov_block_3(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)</code>	245
10.44.2.4	<code>asp_solver_bdcsr_krylov_block_4(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)</code>	245



10.44.2.5 fasp_solver_bdcsl_krylov_sweeping(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, INT NumLayers, block_dCSRmat *Ai, dCSRmat *local_A, iverector *local_index) . . . . .	246
10.45 itsolver_bsr.c File Reference . . . . .	247
10.45.1 Detailed Description . . . . .	247
10.45.2 Function Documentation . . . . .	247
10.45.2.1 fasp_solver_dbsr_itsolver(dBSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam) . . . . .	247
10.45.2.2 fasp_solver_dbsr_krylov(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	248
10.45.2.3 fasp_solver_dbsr_krylov_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam) . . . . .	248
10.45.2.4 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) . . . . .	249
10.45.2.5 fasp_solver_dbsr_krylov_diag(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam) . . . . .	249
10.45.2.6 fasp_solver_dbsr_krylov_ilu(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam) . . . . .	250
10.45.2.7 fasp_solver_dbsr_krylov_nk_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, const INT nk_dim, dvector *nk) . . . . .	250
10.46 itsolver_csr.c File Reference . . . . .	251
10.46.1 Detailed Description . . . . .	252
10.46.2 Function Documentation . . . . .	252
10.46.2.1 fasp_solver_dcsr_itsolver(dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam) . . . . .	252
10.46.2.2 fasp_solver_dcsr_krylov(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	252
10.46.2.3 fasp_solver_dcsr_krylov_amg(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam) . . . . .	253
10.46.2.4 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) . . . . .	253
10.46.2.5 fasp_solver_dcsr_krylov_diag(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam) . . . . .	254
10.46.2.6 fasp_solver_dcsr_krylov_ilu(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam) . . . . .	254
10.46.2.7 fasp_solver_dcsr_krylov_ilu_M(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam, dCSRmat *M) . . . . .	255
10.46.2.8 fasp_solver_dcsr_krylov_Schwarz(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, Schwarz_param *schparam) . . . . .	256
10.47 itsolver_mf.c File Reference . . . . .	257
10.47.1 Detailed Description . . . . .	257
10.47.2 Function Documentation . . . . .	258

10.47.2.1 fasp_solver_itsolver(mxv_matfree *mf, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	258
10.47.2.2 fasp_solver_itsolver_init(INT matrix_format, mxv_matfree *mf, void *A)	259
10.47.2.3 fasp_solver_krylov(mxv_matfree *mf, dvector *b, dvector *x, itsolver_param *itparam)	259
10.48 itsolver_str.c File Reference	260
10.48.1 Detailed Description	260
10.48.2 Function Documentation	261
10.48.2.1 fasp_solver_dstr_itsolver(dSTRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	261
10.48.2.2 fasp_solver_dstr_krylov(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	261
10.48.2.3 fasp_solver_dstr_krylov_blockgs(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ivector *neigh, ivector *order)	262
10.48.2.4 fasp_solver_dstr_krylov_diag(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	263
10.48.2.5 fasp_solver_dstr_krylov_ilu(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)	263
10.49 lu.c File Reference	264
10.49.1 Detailed Description	264
10.49.2 Function Documentation	264
10.49.2.1 fasp_smat_lu_decomp(REAL *A, INT pivot[], const INT n)	264
10.49.2.2 fasp_smat_lu_solve(REAL *A, REAL b[], INT pivot[], REAL x[], const INT n)	265
10.50 memory.c File Reference	266
10.50.1 Detailed Description	266
10.50.2 Function Documentation	267
10.50.2.1 fasp_mem_calloc(LONGLONG size, INT type)	267
10.50.2.2 fasp_mem_check(void *ptr, const char *message, INT ERR)	267
10.50.2.3 fasp_mem_dcsr_check(dCSRmat *A)	267
10.50.2.4 fasp_mem_free(void *mem)	268
10.50.2.5 fasp_mem_iludata_check(ILU_data *iludata)	268
10.50.2.6 fasp_mem_realloc(void *oldmem, LONGLONG tsize)	269
10.50.2.7 fasp_mem_usage()	269
10.50.3 Variable Documentation	269
10.50.3.1 total_alloc_count	269
10.50.3.2 total_alloc_mem	270
10.51 message.c File Reference	270
10.51.1 Detailed Description	270
10.51.2 Function Documentation	270
10.51.2.1 fasp_chkerr(const SHORT status, const char *fname)	270

10.51.2.2	<code>print_amgcomplexity(AMG_data *mgl, const SHORT prtlvl)</code>	271
10.51.2.3	<code>print_amgcomplexity_bsr(AMG_data_bsr *mgl, const SHORT prtlvl)</code>	271
10.51.2.4	<code>print_cputime(const char *message, const REAL cputime)</code>	271
10.51.2.5	<code>print_itinfo(const INT prtlvl, const INT stop_type, const INT iter, const REAL relres, const REAL absres, const REAL factor)</code>	272
10.51.2.6	<code>print_message(const INT prtlvl, const char *message)</code>	272
10.52	<code>mgcycle.c</code> File Reference	273
10.52.1	Detailed Description	273
10.52.2	Function Documentation	273
10.52.2.1	<code>fasp_solver_mgcycle(AMG_data *mgl, AMG_param *param)</code>	273
10.52.2.2	<code>fasp_solver_mgcycle_bsr(AMG_data_bsr *mgl, AMG_param *param)</code>	274
10.53	<code>mgrecur.c</code> File Reference	274
10.53.1	Detailed Description	274
10.53.2	Function Documentation	274
10.53.2.1	<code>fasp_solver_mgrecur(AMG_data *mgl, AMG_param *param, INT level)</code>	274
10.54	<code>ordering.c</code> File Reference	275
10.54.1	Detailed Description	276
10.54.2	Function Documentation	276
10.54.2.1	<code>fasp_aux_dQuickSort(REAL *a, INT left, INT right)</code>	276
10.54.2.2	<code>fasp_aux_dQuickSortIndex(REAL *a, INT left, INT right, INT *index)</code>	276
10.54.2.3	<code>fasp_aux_iQuickSort(INT *a, INT left, INT right)</code>	277
10.54.2.4	<code>fasp_aux_iQuickSortIndex(INT *a, INT left, INT right, INT *index)</code>	278
10.54.2.5	<code>fasp_aux_merge(INT numbers[], INT work[], INT left, INT mid, INT right)</code>	278
10.54.2.6	<code>fasp_aux_msort(INT numbers[], INT work[], INT left, INT right)</code>	279
10.54.2.7	<code>fasp_aux_unique(INT numbers[], const INT size)</code>	279
10.54.2.8	<code>fasp_BinarySearch(INT *list, const INT value, const INT nlist)</code>	280
10.54.2.9	<code>fasp_dcsr_CMK_order(const dCSRmat *A, INT *order, INT *oindex)</code>	280
10.54.2.10	<code>fasp_dcsr_RCMK_order(const dCSRmat *A, INT *order, INT *oindex, INT *rorder)</code>	281
10.55	<code>parameters.c</code> File Reference	281
10.55.1	Detailed Description	282
10.55.2	Function Documentation	282
10.55.2.1	<code>fasp_param_amg_init(AMG_param *amgparam)</code>	282
10.55.2.2	<code>fasp_param_amg_print(AMG_param *param)</code>	283
10.55.2.3	<code>fasp_param_amg_set(AMG_param *param, input_param *iniparam)</code>	283
10.55.2.4	<code>fasp_param_amg_to_prec(precond_data *pcdata, AMG_param *amgparam)</code>	283
10.55.2.5	<code>fasp_param_amg_to_prec_bsr(precond_data_bsr *pcdata, AMG_param *amgparam)</code>	284
10.55.2.6	<code>fasp_param_ilu_init(ILU_param *iluparam)</code>	284

10.55.2.7 fasp_param_ilu_print(ILU_param *param) . . . . .	284
10.55.2.8 fasp_param_ilu_set(ILU_param *iluparam, input_param *iniparam) . . . . .	285
10.55.2.9 fasp_param_init(input_param *iniparam, itsolver_param *itsparam, AMG_param *amgparam, ILU_param *iluparam, Schwarz_param *schparam) . . . . .	285
10.55.2.10 fasp_param_input_init(input_param *iniparam) . . . . .	286
10.55.2.11 fasp_param_prec_to_amg(AMG_param *amgparam, precondition_data *pcdata) . . . . .	286
10.55.2.12 fasp_param_prec_to_amg_bsr(AMG_param *amgparam, precondition_data_bsr *pcdata) . . . . .	286
10.55.2.13 fasp_param_Schwarz_init(Schwarz_param *schparam) . . . . .	287
10.55.2.14 fasp_param_Schwarz_print(Schwarz_param *param) . . . . .	288
10.55.2.15 fasp_param_Schwarz_set(Schwarz_param *schparam, input_param *iniparam) . . . . .	288
10.55.2.16 fasp_param_set(int argc, const char *argv[], input_param *iniparam) . . . . .	288
10.55.2.17 fasp_param_solver_init(itsolver_param *itsparam) . . . . .	289
10.55.2.18 fasp_param_solver_print(itsolver_param *param) . . . . .	289
10.55.2.19 fasp_param_solver_set(itsolver_param *itsparam, input_param *iniparam) . . . . .	289
10.56 pbcgs.c File Reference . . . . .	290
10.56.1 Detailed Description . . . . .	290
10.56.2 Function Documentation . . . . .	291
10.56.2.1 fasp_solver_bdcsl_pbcgs(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	291
10.56.2.2 fasp_solver_dbsl_pbcgs(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	292
10.56.2.3 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) . . . . .	293
10.56.2.4 fasp_solver_dstr_pbcgs(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const R← EAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	293
10.57 pbcgs_mf.c File Reference . . . . .	294
10.57.1 Detailed Description . . . . .	294
10.57.2 Function Documentation . . . . .	295
10.57.2.1 fasp_solver_pbcgs(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const R← EAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	295
10.58 pcg.c File Reference . . . . .	296
10.58.1 Detailed Description . . . . .	297
10.58.2 Function Documentation . . . . .	298
10.58.2.1 fasp_solver_bdcsl_pcg(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	298
10.58.2.2 fasp_solver_dbsl_pcg(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const R← EAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	299
10.58.2.3 fasp_solver_dcsr_pcg(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const R← EAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl) . . . . .	300

10.58.2.4 fasp_solver_dstr_pcg(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	301
10.59pcg_mf.c File Reference	302
10.59.1 Detailed Description	302
10.59.2 Function Documentation	303
10.59.2.1 fasp_solver_pcg(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	303
10.60pgcg.c File Reference	304
10.60.1 Detailed Description	304
10.60.2 Function Documentation	304
10.60.2.1 fasp_solver_dcsr_pcg(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	304
10.61pgcg_mf.c File Reference	305
10.61.1 Detailed Description	305
10.61.2 Function Documentation	305
10.61.2.1 fasp_solver_pcg(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	305
10.62pgcr.c File Reference	306
10.62.1 Detailed Description	306
10.62.2 Function Documentation	307
10.62.2.1 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 prtvl)	307
10.62.2.2 fasp_solver_dcsr_pgcr1(dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	308
10.63pgmres.c File Reference	309
10.63.1 Detailed Description	309
10.63.2 Function Documentation	309
10.63.2.1 fasp_solver_bdcsr_pgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	309
10.63.2.2 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 prtvl)	310
10.63.2.3 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 prtvl)	311
10.63.2.4 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 prtvl)	311
10.64pgmres_mf.c File Reference	312

10.64.1 Detailed Description	312
10.64.2 Function Documentation	313
10.64.2.1 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 prtvl)	313
10.65pminres.c File Reference	313
10.65.1 Detailed Description	314
10.65.2 Function Documentation	315
10.65.2.1 fasp_solver_bdcsr_pminres(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	315
10.65.2.2 fasp_solver_dcsr_pminres(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	315
10.65.2.3 fasp_solver_dstr_pminres(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	316
10.66pminres_mf.c File Reference	317
10.66.1 Detailed Description	317
10.66.2 Function Documentation	318
10.66.2.1 fasp_solver_pminres(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	318
10.67precond_bcsr.c File Reference	319
10.67.1 Detailed Description	319
10.67.2 Function Documentation	320
10.67.2.1 fasp_precond_block_diag_3(REAL *r, REAL *z, void *data)	320
10.67.2.2 fasp_precond_block_diag_3_amg(REAL *r, REAL *z, void *data)	321
10.67.2.3 fasp_precond_block_diag_4(REAL *r, REAL *z, void *data)	321
10.67.2.4 fasp_precond_block_lower_3(REAL *r, REAL *z, void *data)	322
10.67.2.5 fasp_precond_block_lower_3_amg(REAL *r, REAL *z, void *data)	323
10.67.2.6 fasp_precond_block_lower_4(REAL *r, REAL *z, void *data)	323
10.67.2.7 fasp_precond_block_SGS_3(REAL *r, REAL *z, void *data)	324
10.67.2.8 fasp_precond_block_SGS_3_amg(REAL *r, REAL *z, void *data)	325
10.67.2.9 fasp_precond_block_upper_3(REAL *r, REAL *z, void *data)	325
10.67.2.10 fasp_precond_block_upper_3_amg(REAL *r, REAL *z, void *data)	326
10.67.2.11 fasp_precond_sweeping(REAL *r, REAL *z, void *data)	327
10.68precond_bsr.c File Reference	327
10.68.1 Detailed Description	328
10.68.2 Function Documentation	328
10.68.2.1 fasp_precond_dbsr_amg(REAL *r, REAL *z, void *data)	328
10.68.2.2 fasp_precond_dbsr_amg_nk(REAL *r, REAL *z, void *data)	328
10.68.2.3 fasp_precond_dbsr_diag(REAL *r, REAL *z, void *data)	329

10.68.2.4 fasp_precond_dbsr_diag_nc2(REAL *r, REAL *z, void *data) . . . . .	329
10.68.2.5 fasp_precond_dbsr_diag_nc3(REAL *r, REAL *z, void *data) . . . . .	330
10.68.2.6 fasp_precond_dbsr_diag_nc5(REAL *r, REAL *z, void *data) . . . . .	331
10.68.2.7 fasp_precond_dbsr_diag_nc7(REAL *r, REAL *z, void *data) . . . . .	332
10.68.2.8 fasp_precond_dbsr_ilu(REAL *r, REAL *z, void *data) . . . . .	333
10.68.2.9 fasp_precond_dbsr_nl_amli(REAL *r, REAL *z, void *data) . . . . .	334
10.69precond_csr.c File Reference . . . . .	334
10.69.1 Detailed Description . . . . .	335
10.69.2 Function Documentation . . . . .	335
10.69.2.1 fasp_precond_amg(REAL *r, REAL *z, void *data) . . . . .	335
10.69.2.2 fasp_precond_amg_nk(REAL *r, REAL *z, void *data) . . . . .	336
10.69.2.3 fasp_precond_amli(REAL *r, REAL *z, void *data) . . . . .	336
10.69.2.4 fasp_precond_diag(REAL *r, REAL *z, void *data) . . . . .	336
10.69.2.5 fasp_precond_famg(REAL *r, REAL *z, void *data) . . . . .	337
10.69.2.6 fasp_precond_free(const SHORT precondition_type, precondition *pc) . . . . .	337
10.69.2.7 fasp_precond_ilu(REAL *r, REAL *z, void *data) . . . . .	338
10.69.2.8 fasp_precond_ilu_backward(REAL *r, REAL *z, void *data) . . . . .	338
10.69.2.9 fasp_precond_ilu_forward(REAL *r, REAL *z, void *data) . . . . .	338
10.69.2.10 fasp_precond_nl_amli(REAL *r, REAL *z, void *data) . . . . .	339
10.69.2.11 fasp_precond_Schwarz(REAL *r, REAL *z, void *data) . . . . .	339
10.69.2.12 fasp_precond_setup(const SHORT precondition_type, AMG_param *amgparam, ILU_↵ param *iluparam, dCSRmat *A) . . . . .	339
10.70precond_str.c File Reference . . . . .	340
10.70.1 Detailed Description . . . . .	341
10.70.2 Function Documentation . . . . .	341
10.70.2.1 fasp_precond_dstr_blockgs(REAL *r, REAL *z, void *data) . . . . .	341
10.70.2.2 fasp_precond_dstr_diag(REAL *r, REAL *z, void *data) . . . . .	341
10.70.2.3 fasp_precond_dstr_ilu0(REAL *r, REAL *z, void *data) . . . . .	341
10.70.2.4 fasp_precond_dstr_ilu0_backward(REAL *r, REAL *z, void *data) . . . . .	342
10.70.2.5 fasp_precond_dstr_ilu0_forward(REAL *r, REAL *z, void *data) . . . . .	342
10.70.2.6 fasp_precond_dstr_ilu1(REAL *r, REAL *z, void *data) . . . . .	343
10.70.2.7 fasp_precond_dstr_ilu1_backward(REAL *r, REAL *z, void *data) . . . . .	344
10.70.2.8 fasp_precond_dstr_ilu1_forward(REAL *r, REAL *z, void *data) . . . . .	344
10.71pvfgmres.c File Reference . . . . .	345
10.71.1 Detailed Description . . . . .	345
10.71.2 Function Documentation . . . . .	345

10.71.2.1 fasp_solver_bdcslr_pvfgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	345
10.71.2.2 fasp_solver_dbsl_r_pvfgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	346
10.71.2.3 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 prtvl)	347
10.72pvfgmres_mf.c File Reference	347
10.72.1 Detailed Description	348
10.72.2 Function Documentation	348
10.72.2.1 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 prtvl)	348
10.73pvgmres.c File Reference	349
10.73.1 Detailed Description	349
10.73.2 Function Documentation	349
10.73.2.1 fasp_solver_bdcslr_pvgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	349
10.73.2.2 fasp_solver_dbsl_r_pvgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	350
10.73.2.3 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 prtvl)	351
10.73.2.4 fasp_solver_dstl_r_pvgmres(dSTRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	352
10.74pvgmres_mf.c File Reference	353
10.74.1 Detailed Description	353
10.74.2 Function Documentation	353
10.74.2.1 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 prtvl)	353
10.75quadrature.c File Reference	354
10.75.1 Detailed Description	354
10.75.2 Function Documentation	354
10.75.2.1 fasp_gauss2d(const INT num_qp, const INT ncoor, REAL(*gauss)[3])	354
10.75.2.2 fasp_quad2d(const INT num_qp, const INT ncoor, REAL(*quad)[3])	355
10.76rap.c File Reference	355
10.76.1 Detailed Description	356



10.76.2 Function Documentation	356
10.76.2.1 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)	356
10.77schwarz_setup.c File Reference	356
10.77.1 Detailed Description	357
10.77.2 Function Documentation	357
10.77.2.1 fasp_dcsr_Schwarz_backward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)	357
10.77.2.2 fasp_dcsr_Schwarz_forward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)	357
10.77.2.3 fasp_Schwarz_get_block_matrix(Schwarz_data *Schwarz, INT nblk, INT *iblock, I← NT *jblock, INT *mask)	358
10.77.2.4 fasp_Schwarz_setup(Schwarz_data *Schwarz, Schwarz_param *param)	358
10.78smat.c File Reference	359
10.78.1 Detailed Description	359
10.78.2 Macro Definition Documentation	360
10.78.2.1 SWAP	360
10.78.3 Function Documentation	360
10.78.3.1 fasp_blas_smat_inv(REAL *a, const INT n)	360
10.78.3.2 fasp_blas_smat_inv_nc(REAL *a, const INT n)	360
10.78.3.3 fasp_blas_smat_inv_nc2(REAL *a)	360
10.78.3.4 fasp_blas_smat_inv_nc3(REAL *a)	361
10.78.3.5 fasp_blas_smat_inv_nc4(REAL *a)	361
10.78.3.6 fasp_blas_smat_inv_nc5(REAL *a)	361
10.78.3.7 fasp_blas_smat_inv_nc7(REAL *a)	362
10.78.3.8 fasp_blas_smat_invp_nc(REAL *a, const INT n)	362
10.78.3.9 fasp_blas_smat_Linfinity(REAL *A, const INT n)	363
10.78.3.10fasp_iden_free(idenmat *A)	363
10.78.3.11fasp_smat_identity(REAL *a, const INT n, const INT n2)	363
10.78.3.12fasp_smat_identity_nc2(REAL *a)	364
10.78.3.13fasp_smat_identity_nc3(REAL *a)	365
10.78.3.14fasp_smat_identity_nc5(REAL *a)	365
10.78.3.15fasp_smat_identity_nc7(REAL *a)	365
10.79smoother_bsr.c File Reference	366
10.79.1 Detailed Description	367
10.79.2 Function Documentation	367
10.79.2.1 fasp_smoother_dbsr_gs(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark)	367

10.79.2.2 fasp_smoother_dbsr_gs1(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv) . . . . .	367
10.79.2.3 fasp_smoother_dbsr_gs_ascend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	368
10.79.2.4 fasp_smoother_dbsr_gs_ascend1(dBSRmat *A, dvector *b, dvector *u) . . . . .	368
10.79.2.5 fasp_smoother_dbsr_gs_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	369
10.79.2.6 fasp_smoother_dbsr_gs_descend1(dBSRmat *A, dvector *b, dvector *u) . . . . .	369
10.79.2.7 fasp_smoother_dbsr_gs_order1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark) . . . . .	370
10.79.2.8 fasp_smoother_dbsr_gs_order2(dBSRmat *A, dvector *b, dvector *u, INT *mark, REAL *work) . . . . .	370
10.79.2.9 fasp_smoother_dbsr_ilu(dBSRmat *A, dvector *b, dvector *x, void *data) . . . . .	371
10.79.2.10 fasp_smoother_dbsr_jacobi(dBSRmat *A, dvector *b, dvector *u) . . . . .	371
10.79.2.11 fasp_smoother_dbsr_jacobi1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv) . . . . .	372
10.79.2.12 fasp_smoother_dbsr_jacobi_setup(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv) . . . . .	373
10.79.2.13 fasp_smoother_dbsr_sor(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL weight) . . . . .	373
10.79.2.14 fasp_smoother_dbsr_sor1(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv, REAL weight) . . . . .	374
10.79.2.15 fasp_smoother_dbsr_sor_ascend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight) . . . . .	374
10.79.2.16 fasp_smoother_dbsr_sor_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight) . . . . .	375
10.79.2.17 fasp_smoother_dbsr_sor_order(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight) . . . . .	375
10.80 smoother_csr.c File Reference . . . . .	376
10.80.1 Detailed Description . . . . .	377
10.80.2 Function Documentation . . . . .	377
10.80.2.1 fasp_smoother_dcsr_gs(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L) . . . . .	377
10.80.2.2 fasp_smoother_dcsr_gs_cf(dvector *u, dCSRmat *A, dvector *b, INT L, INT *mark, const INT order) . . . . .	377
10.80.2.3 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) . . . . .	378
10.80.2.4 fasp_smoother_dcsr_ilu(dCSRmat *A, dvector *b, dvector *x, void *data) . . . . .	378
10.80.2.5 fasp_smoother_dcsr_jacobi(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L) . . . . .	379
10.80.2.6 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) . . . . .	379
10.80.2.7 fasp_smoother_dcsr_L1diag(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L) . . . . .	380
10.80.2.8 fasp_smoother_dcsr_sgs(dvector *u, dCSRmat *A, dvector *b, INT L) . . . . .	380

10.80.2.9 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) . . . . .	381
10.80.2.10 fasp_smoother_dcsr_sor_cf(dvector *u, dCSRmat *A, dvector *b, INT L, const REAL w, INT *mark, const INT order) . . . . .	381
10.81 smoother_csr_cr.c File Reference . . . . .	382
10.81.1 Detailed Description . . . . .	382
10.81.2 Function Documentation . . . . .	382
10.81.2.1 fasp_smoother_dcsr_gscr(INT pt, INT n, REAL *u, INT *ia, INT *ja, REAL *a, REAL *b, INT L, INT *CF) . . . . .	382
10.82 smoother_csr_poly.c File Reference . . . . .	383
10.82.1 Detailed Description . . . . .	383
10.82.2 Function Documentation . . . . .	384
10.82.2.1 fasp_smoother_dcsr_poly(dCSRmat *Amat, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L) . . . . .	384
10.82.2.2 fasp_smoother_dcsr_poly_old(dCSRmat *Amat, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L) . . . . .	384
10.83 smoother_str.c File Reference . . . . .	384
10.83.1 Detailed Description . . . . .	385
10.83.2 Function Documentation . . . . .	386
10.83.2.1 fasp_generate_diaginv_block(dSTRmat *A, ivector *neigh, dvector *diaginv, ivector *pivot) . . . . .	386
10.83.2.2 fasp_smoother_dstr_gs(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark) . . . . .	386
10.83.2.3 fasp_smoother_dstr_gs1(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, REAL *diaginv) . . . . .	386
10.83.2.4 fasp_smoother_dstr_gs_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv) . . . . .	387
10.83.2.5 fasp_smoother_dstr_gs_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, const INT order) . . . . .	387
10.83.2.6 fasp_smoother_dstr_gs_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv) . . . . .	388
10.83.2.7 fasp_smoother_dstr_gs_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark) . . . . .	388
10.83.2.8 fasp_smoother_dstr_jacobi(dSTRmat *A, dvector *b, dvector *u) . . . . .	389
10.83.2.9 fasp_smoother_dstr_jacobi1(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv) . . . . .	389
10.83.2.10 fasp_smoother_dstr_schwarz(dSTRmat *A, dvector *b, dvector *u, dvector *diaginv, ivector *pivot, ivector *neigh, ivector *order) . . . . .	390
10.83.2.11 fasp_smoother_dstr_sor(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, const REAL weight) . . . . .	390
10.83.2.12 fasp_smoother_dstr_sor1(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, REAL *diaginv, const REAL weight) . . . . .	391
10.83.2.13 fasp_smoother_dstr_sor_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight) . . . . .	391

10.83.2.14	<code>asp_smoother_dstr_sor_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, const INT order, const REAL weight)</code>	391
10.83.2.15	<code>asp_smoother_dstr_sor_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)</code>	392
10.83.2.16	<code>asp_smoother_dstr_sor_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight)</code>	392
10.84	<code>sparse_block.c</code> File Reference	393
10.84.1	Detailed Description	393
10.84.2	Function Documentation	394
10.84.2.1	<code>asp_bdcsr_free(block_dCSRmat *A)</code>	394
10.84.2.2	<code>asp_dbsr_getblk(dBSRmat *A, INT *Is, INT *Js, const INT m, const INT n, dBSRmat *B)</code>	395
10.84.2.3	<code>asp_dbsr_getblk_dcsr(dBSRmat *A)</code>	395
10.84.2.4	<code>asp_dbsr_Linfinity_dcsr(dBSRmat *A)</code>	396
10.84.2.5	<code>asp_dcsr_getblk(dCSRmat *A, INT *Is, INT *Js, const INT m, const INT n, dCSRmat *B)</code>	396
10.85	<code>sparse_bsr.c</code> File Reference	397
10.85.1	Detailed Description	398
10.85.2	Function Documentation	398
10.85.2.1	<code>asp_dbsr_alloc(const INT ROW, const INT COL, const INT NNZ, const INT nb, const INT storage_manner, dBSRmat *A)</code>	398
10.85.2.2	<code>asp_dbsr_cp(dBSRmat *A, dBSRmat *B)</code>	398
10.85.2.3	<code>asp_dbsr_create(const INT ROW, const INT COL, const INT NNZ, const INT nb, const INT storage_manner)</code>	399
10.85.2.4	<code>asp_dbsr_diaginv(dBSRmat *A)</code>	399
10.85.2.5	<code>asp_dbsr_diaginv2(dBSRmat *A, REAL *diaginv)</code>	400
10.85.2.6	<code>asp_dbsr_diaginv3(dBSRmat *A, REAL *diaginv)</code>	400
10.85.2.7	<code>asp_dbsr_diaginv4(dBSRmat *A, REAL *diaginv)</code>	401
10.85.2.8	<code>asp_dbsr_diagLU(dBSRmat *A, REAL *DL, REAL *DU)</code>	401
10.85.2.9	<code>asp_dbsr_diagLU2(dBSRmat *A, REAL *DL, REAL *DU)</code>	402
10.85.2.10	<code>asp_dbsr_diagpref(dBSRmat *A)</code>	402
10.85.2.11	<code>asp_dbsr_free(dBSRmat *A)</code>	403
10.85.2.12	<code>asp_dbsr_getdiag(INT n, dBSRmat *A, REAL *diag)</code>	403
10.85.2.13	<code>asp_dbsr_getdiaginv(dBSRmat *A)</code>	404
10.85.2.14	<code>asp_dbsr_null(dBSRmat *A)</code>	404
10.85.2.15	<code>asp_dbsr_trans(dBSRmat *A, dBSRmat *AT)</code>	405
10.86	<code>sparse_coo.c</code> File Reference	405
10.86.1	Detailed Description	405
10.86.2	Function Documentation	405

10.86.2.1	<code>fasp_dcoo_alloc(const INT m, const INT n, const INT nnz, dCOOmat *A)</code>	405
10.86.2.2	<code>fasp_dcoo_create(const INT m, const INT n, const INT nnz)</code>	406
10.86.2.3	<code>fasp_dcoo_free(dCOOmat *A)</code>	406
10.86.2.4	<code>fasp_dcoo_shift(dCOOmat *A, const INT offset)</code>	407
10.87	<code>sparse_csr.c</code> File Reference	407
10.87.1	Detailed Description	408
10.87.2	Function Documentation	408
10.87.2.1	<code>fasp_dcsr_alloc(const INT m, const INT n, const INT nnz, dCSRmat *A)</code>	408
10.87.2.2	<code>fasp_dcsr_compress(dCSRmat *A, dCSRmat *B, REAL dtol)</code>	409
10.87.2.3	<code>fasp_dcsr_compress_inplace(dCSRmat *A, REAL dtol)</code>	409
10.87.2.4	<code>fasp_dcsr_cp(dCSRmat *A, dCSRmat *B)</code>	410
10.87.2.5	<code>fasp_dcsr_create(const INT m, const INT n, const INT nnz)</code>	410
10.87.2.6	<code>fasp_dcsr_diagpref(dCSRmat *A)</code>	411
10.87.2.7	<code>fasp_dcsr_free(dCSRmat *A)</code>	412
10.87.2.8	<code>fasp_dcsr_getcol(const INT n, dCSRmat *A, REAL *col)</code>	412
10.87.2.9	<code>fasp_dcsr_getdiag(INT n, dCSRmat *A, dvector *diag)</code>	413
10.87.2.10	<code>fasp_dcsr_multicoloring(dCSRmat *A, INT *flags, INT *groups)</code>	413
10.87.2.11	<code>fasp_dcsr_null(dCSRmat *A)</code>	414
10.87.2.12	<code>fasp_dcsr_perm(dCSRmat *A, INT *P)</code>	414
10.87.2.13	<code>fasp_dcsr_permz(dCSRmat *A, INT *p)</code>	414
10.87.2.14	<code>fasp_dcsr_regdiag(dCSRmat *A, REAL value)</code>	415
10.87.2.15	<code>fasp_dcsr_shift(dCSRmat *A, INT offset)</code>	415
10.87.2.16	<code>fasp_dcsr_sort(dCSRmat *A)</code>	416
10.87.2.17	<code>fasp_dcsr_sortz(dCSRmat *A, const SHORT isym)</code>	416
10.87.2.18	<code>fasp_dcsr_symdiagscale(dCSRmat *A, dvector *diag)</code>	417
10.87.2.19	<code>fasp_dcsr_sympat(dCSRmat *A)</code>	417
10.87.2.20	<code>fasp_dcsr_trans(dCSRmat *A, dCSRmat *AT)</code>	417
10.87.2.21	<code>fasp_dcsr_transz(dCSRmat *A, INT *p, dCSRmat *AT)</code>	418
10.87.2.22	<code>fasp_icsr_cp(iCSRmat *A, iCSRmat *B)</code>	418
10.87.2.23	<code>fasp_icsr_create(const INT m, const INT n, const INT nnz)</code>	419
10.87.2.24	<code>fasp_icsr_free(iCSRmat *A)</code>	419
10.87.2.25	<code>fasp_icsr_null(iCSRmat *A)</code>	420
10.87.2.26	<code>fasp_icsr_trans(iCSRmat *A, iCSRmat *AT)</code>	421
10.88	<code>sparse_csrl.c</code> File Reference	421
10.88.1	Detailed Description	422
10.88.2	Function Documentation	422
10.88.2.1	<code>fasp_dcsrl_create(const INT num_rows, const INT num_cols, const INT num_nonzeros)</code>	422

10.88.2.2 fasp_dcsr_free(dCSRmat *A)	422
10.89 sparse_str.c File Reference	422
10.89.1 Detailed Description	423
10.89.2 Function Documentation	423
10.89.2.1 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)	423
10.89.2.2 fasp_dstr_cp(dSTRmat *A, dSTRmat *A1)	424
10.89.2.3 fasp_dstr_create(const INT nx, const INT ny, const INT nz, const INT nc, const INT nband, INT *offsets)	425
10.89.2.4 fasp_dstr_free(dSTRmat *A)	425
10.89.2.5 fasp_dstr_null(dSTRmat *A)	426
10.90 sparse_util.c File Reference	426
10.90.1 Detailed Description	427
10.90.2 Function Documentation	427
10.90.2.1 fasp_sparse_aat_(INT *ia, INT *ja, REAL *a, INT *na, INT *ma, INT *iat, INT *jat, REAL *at)	427
10.90.2.2 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)	428
10.90.2.3 fasp_sparse_abybms_(INT *ia, INT *ja, INT *ib, INT *jb, INT *nap, INT *map, INT *mbp, INT *ic, INT *jc)	428
10.90.2.4 fasp_sparse_aplbms_(INT *ia, INT *ja, INT *ib, INT *jb, INT *nab, INT *mab, INT *ic, INT *jc)	429
10.90.2.5 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)	429
10.90.2.6 fasp_sparse_iit_(INT *ia, INT *ja, INT *na, INT *ma, INT *iat, INT *jat)	429
10.90.2.7 fasp_sparse_MIS(dCSRmat *A)	430
10.90.2.8 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)	430
10.90.2.9 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)	431
10.90.2.10 fasp_sparse_wta_(INT *jw, REAL *w, INT *ia, INT *ja, REAL *a, INT *nwp, INT *map, INT *jv, REAL *v, INT *nvp)	432
10.90.2.11 fasp_sparse_wtams_(INT *jw, INT *ia, INT *ja, INT *nwp, INT *map, INT *jv, INT *nvp, INT *icp)	432
10.90.2.12 fasp_sparse_ytx_(INT *jy, REAL *y, INT *jx, REAL *x, INT *nyp, INT *nxp, INT *icp, REAL *s)	432
10.90.2.13 fasp_sparse_ytxbig_(INT *jy, REAL *y, INT *nyp, REAL *x, REAL *s)	433
10.91 spbcgs.c File Reference	433
10.91.1 Detailed Description	434
10.91.2 Function Documentation	435

10.91.2.1 fasp_solver_bdcslr_spgcgs(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	435
10.91.2.2 fasp_solver_dbsr_spgcgs(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	435
10.91.2.3 fasp_solver_dcsr_spgcgs(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	436
10.91.2.4 fasp_solver_dstr_spgcgs(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	437
10.92spcg.c File Reference	438
10.92.1 Detailed Description	439
10.92.2 Function Documentation	440
10.92.2.1 fasp_solver_bdcslr_spgc(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	440
10.92.2.2 fasp_solver_dcsr_spgc(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	440
10.92.2.3 fasp_solver_dstr_spgc(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	441
10.93spgmres.c File Reference	442
10.93.1 Detailed Description	442
10.93.2 Function Documentation	442
10.93.2.1 fasp_solver_bdcslr_spgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	442
10.93.2.2 fasp_solver_dbsr_spgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	443
10.93.2.3 fasp_solver_dcsr_spgmres(dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	444
10.93.2.4 fasp_solver_dstr_spgmres(dSTRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	445
10.94spminres.c File Reference	446
10.94.1 Detailed Description	446
10.94.2 Function Documentation	447
10.94.2.1 fasp_solver_bdcslr_spminres(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	447
10.94.2.2 fasp_solver_dcsr_spminres(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	448
10.94.2.3 fasp_solver_dstr_spminres(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	449
10.95spvgmres.c File Reference	450
10.95.1 Detailed Description	451

10.95.2 Function Documentation	451
10.95.2.1 fasp_solver_bdcslr_spvgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	451
10.95.2.2 fasp_solver_dbsr_spvgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	451
10.95.2.3 fasp_solver_dcsr_spvgmres(dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	452
10.95.2.4 fasp_solver_dstr_spvgmres(dSTRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtvl)	453
10.96threads.c File Reference	453
10.96.1 Detailed Description	454
10.96.2 Function Documentation	454
10.96.2.1 FASP_GET_START_END(INT procid, INT nprocs, INT n, INT *start, INT *end)	454
10.96.2.2 fasp_set_GS_threads(INT mythreads, INT its)	454
10.96.3 Variable Documentation	454
10.96.3.1 THDs_AMG_GS	454
10.96.3.2 THDs_CPR_gGS	455
10.96.3.3 THDs_CPR_IGS	455
10.97timing.c File Reference	455
10.97.1 Detailed Description	455
10.97.2 Function Documentation	455
10.97.2.1 fasp_gettime(REAL *time)	455
10.98vec.c File Reference	456
10.98.1 Detailed Description	456
10.98.2 Function Documentation	457
10.98.2.1 fasp_dvec_alloc(const INT m, dvector *u)	457
10.98.2.2 fasp_dvec_cp(dvector *x, dvector *y)	458
10.98.2.3 fasp_dvec_create(const INT m)	458
10.98.2.4 fasp_dvec_free(dvector *u)	459
10.98.2.5 fasp_dvec_isnan(dvector *u)	460
10.98.2.6 fasp_dvec_maxdiff(dvector *x, dvector *y)	460
10.98.2.7 fasp_dvec_null(dvector *x)	461
10.98.2.8 fasp_dvec_rand(const INT n, dvector *x)	461
10.98.2.9 fasp_dvec_set(INT n, dvector *x, REAL val)	462
10.98.2.10 fasp_dvec_symdiagscale(dvector *b, dvector *diag)	462



10.98.2.1 fasp_ivec_alloc(const INT m, ivector *u)	462
10.98.2.2 fasp_ivec_create(const INT m)	463
10.98.2.3 fasp_ivec_free(ivector *u)	463
10.98.2.4 fasp_ivec_set(const INT m, ivector *u)	464
10.99 wrapper.c File Reference	464
10.99.1 Detailed Description	464
10.99.2 Function Documentation	465
10.99.2.1 fasp_fwrapper_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	465
10.99.2.2 fasp_fwrapper_krylov_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	465
10.99.2.3 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)	466
10.99.2.4 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)	466
<b>Index</b>	<b>469</b>



# Chapter 1

## Introduction

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

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

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 is 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

For the moment, FASP is still under alpha testing. You need a password to download the package. Sorry about it!

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)
- Zikatanov, Ludmil (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

# Todo List

File [sparse\\_util.c](#)

Remove unwanted functions from this file. –Chensong



## Chapter 7

# Data Structure Index

### 7.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">AMG_data</a>	Data for AMG solvers . . . . .	23
<a href="#">AMG_data_bsr</a>	Data for multigrid levels. (BSR format) . . . . .	24
<a href="#">AMG_param</a>	Parameters for AMG solver . . . . .	26
<a href="#">block_BSR</a>	Block REAL matrix format for reservoir simulation . . . . .	28
<a href="#">block_dCSRmat</a>	Block REAL CSR matrix format . . . . .	28
<a href="#">block_dvector</a>	Block REAL vector structure . . . . .	29
<a href="#">block_iCSRmat</a>	Block INT CSR matrix format . . . . .	29
<a href="#">block_ivector</a>	Block INT vector structure . . . . .	30
<a href="#">block_Reservoir</a>	Block REAL matrix format for reservoir simulation . . . . .	31
<a href="#">dBSRmat</a>	Block sparse row storage matrix of REAL type . . . . .	31
<a href="#">dCOOmat</a>	Sparse matrix of REAL type in COO (or IJ) format . . . . .	32
<a href="#">dCSRLmat</a>	Sparse matrix of REAL type in CSRL format . . . . .	33
<a href="#">dCSRmat</a>	Sparse matrix of REAL type in CSR format . . . . .	34
<a href="#">ddenmat</a>	Dense matrix of REAL type . . . . .	34
<a href="#">dSTRmat</a>	Structure matrix of REAL type . . . . .	35
<a href="#">dvector</a>	Vector with n entries of REAL type . . . . .	36
<a href="#">grid2d</a>	Two dimensional grid data structure . . . . .	36

<a href="#">iCOOmat</a>	Sparse matrix of INT type in COO (or IJ) format . . . . .	38
<a href="#">iCSRmat</a>	Sparse matrix of INT type in CSR format . . . . .	39
<a href="#">idenmat</a>	Dense matrix of INT type . . . . .	40
<a href="#">ILU_data</a>	Data for ILU setup . . . . .	40
<a href="#">ILU_param</a>	Parameters for ILU . . . . .	41
<a href="#">input_param</a>	Input parameters . . . . .	42
<a href="#">itsolver_param</a>	Parameters passed to iterative solvers . . . . .	49
<a href="#">ivector</a>	Vector with n entries of INT type . . . . .	50
<a href="#">Link</a>	Struct for Links . . . . .	51
<a href="#">linked_list</a>	A linked list node . . . . .	51
<a href="#">mallinfo</a>	. . . . .	52
<a href="#">malloc_chunk</a>	. . . . .	52
<a href="#">malloc_params</a>	. . . . .	53
<a href="#">malloc_segment</a>	. . . . .	53
<a href="#">malloc_state</a>	. . . . .	53
<a href="#">malloc_tree_chunk</a>	. . . . .	54
<a href="#">Mumps_data</a>	Parameters for MUMPS interface . . . . .	55
<a href="#">mxv_matfree</a>	Matrix-vector multiplication, replace the actual matrix . . . . .	55
<a href="#">nedmallinfo</a>	. . . . .	55
<a href="#">precond</a>	Preconditioner data and action . . . . .	56
<a href="#">precond_block_data</a>	Data passed to the preconditioner for block preconditioning for <a href="#">block_dCSRmat</a> format . . . . .	56
<a href="#">precond_block_reservoir_data</a>	Data passed to the preconditioner for reservoir simulation problems . . . . .	58
<a href="#">precond_data</a>	Data passed to the preconditioners . . . . .	61
<a href="#">precond_data_bsr</a>	Data passed to the preconditioners . . . . .	63
<a href="#">precond_data_str</a>	Data passed to the preconditioner for <a href="#">dSTRmat</a> matrices . . . . .	64
<a href="#">precond_diagbsr</a>	Data passed to diagonal preconditioner for <a href="#">dBSRmat</a> matrices . . . . .	66
<a href="#">precond_diagstr</a>	Data passed to diagonal preconditioner for <a href="#">dSTRmat</a> matrices . . . . .	66
<a href="#">precond_FASP_blkoi_data</a>	Data passed to the preconditioner for preconditioning reservoir simulation problems . . . . .	67
<a href="#">precond_sweeping_data</a>	Data passed to the preconditioner for sweeping preconditioning . . . . .	71
<a href="#">Schwarz_data</a>	Data for Schwarz methods . . . . .	73



[Schwarz\\_param](#)

Parameters for Schwarz method . . . . .	74
---	----



## Chapter 8

# File Index

### 8.1 File List

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

<a href="#">amg.c</a>	AMG method as an iterative solver (main file)	77
<a href="#">amg_setup_cr.c</a>	Brannick-Falgout compatible relaxation based AMG: SETUP phase	78
<a href="#">amg_setup_rs.c</a>	Ruge-Stuben AMG: SETUP phase	79
<a href="#">amg_setup_sa.c</a>	Smoothed aggregation AMG: SETUP phase	80
<a href="#">amg_setup_ua.c</a>	Unsmoothed aggregation AMG: SETUP phase	82
<a href="#">amg_solve.c</a>	Algebraic multigrid iterations: SOLVE phase	84
<a href="#">amlirecur.c</a>	Abstract AMLI multilevel iteration – recursive version	87
<a href="#">array.c</a>	Simple array operations – init, set, copy, etc	90
<a href="#">blas_array.c</a>	BLAS1 operations for arrays	94
<a href="#">blas_bcsr.c</a>	BLAS2 operations for <a href="#">block_dCSRmat</a> matrices	99
<a href="#">blas_bsr.c</a>	BLAS2 operations for <a href="#">dBSRmat</a> matrices	101
<a href="#">blas_csr.c</a>	BLAS2 operations for <a href="#">dCSRmat</a> matrices	108
<a href="#">blas_csrl.c</a>	BLAS2 operations for <a href="#">dCSRLmat</a> matrices	117
<a href="#">blas_smat.c</a>	BLAS2 operations for <i>small</i> dense matrices	118
<a href="#">blas_str.c</a>	BLAS2 operations for <a href="#">dSTRmat</a> matrices	138
<a href="#">blas_vec.c</a>	BLAS1 operations for vectors	140
<a href="#">checkmat.c</a>	Check matrix properties	144

<a href="#">coarsening_cr.c</a>	
Coarsening with Brannick-Falgout strategy . . . . .	149
<a href="#">coarsening_rs.c</a>	
Coarsening with a modified Ruge-Stuben strategy . . . . .	150
<a href="#">convert.c</a>	
Some utilities for format conversion . . . . .	152
<b>dlmalloc.h</b> . . . . .	??
<a href="#">doxygen.h</a>	
Main page for Doxygen documentation . . . . .	156
<a href="#">eigen.c</a>	
Subroutines for computing the extreme eigenvalues . . . . .	156
<a href="#">famg.c</a>	
Full AMG method as an iterative solver (main file) . . . . .	157
<a href="#">fasp.h</a>	
Main header file for FASP . . . . .	158
<a href="#">fasp_block.h</a>	
Header file for FASP block matrices . . . . .	166
<a href="#">fasp_const.h</a>	
Definition of all kinds of messages, including error messages, solver types, etc . . . . .	169
<a href="#">fmgcycle.c</a>	
Abstract non-recursive full multigrid cycle . . . . .	188
<a href="#">formats.c</a>	
Subroutines for matrix format conversion . . . . .	189
<a href="#">givens.c</a>	
Givens transformation . . . . .	194
<a href="#">gmg_poisson.c</a>	
GMG method as an iterative solver for Poisson Problem . . . . .	195
<a href="#">graphics.c</a>	
Subroutines for graphical output . . . . .	202
<b>hb_io.h</b> . . . . .	??
<a href="#">ilu_setup_bsr.c</a>	
Setup incomplete LU decomposition for <a href="#">dBSRmat</a> matrices . . . . .	205
<a href="#">ilu_setup_csr.c</a>	
Setup incomplete LU decomposition for <a href="#">dCSRmat</a> matrices . . . . .	206
<a href="#">ilu_setup_str.c</a>	
Setup incomplete LU decomposition for <a href="#">dSTRmat</a> matrices . . . . .	207
<a href="#">init.c</a>	
Initialize important data structures . . . . .	208
<a href="#">input.c</a>	
Read input parameters . . . . .	213
<a href="#">interface_mumps.c</a>	
Interface to MUMPS direct solvers . . . . .	214
<a href="#">interface_samg.c</a>	
Interface to SAMG solvers . . . . .	216
<a href="#">interface_superlu.c</a>	
Interface to SuperLU direct solvers . . . . .	217
<a href="#">interface_umfpack.c</a>	
Interface to UMFPACK direct solvers . . . . .	218
<a href="#">interpolation.c</a>	
Interpolation operators for AMG . . . . .	219
<a href="#">interpolation_em.c</a>	
Interpolation operators for AMG based on energy-min . . . . .	221
<a href="#">io.c</a>	
Matrix/vector input/output subroutines . . . . .	222

<a href="#">itsolver_bcsr.c</a>	
Iterative solvers for <a href="#">block_dCSRmat</a> matrices	243
<a href="#">itsolver_bsr.c</a>	
Iterative solvers for <a href="#">dBSRmat</a> matrices	247
<a href="#">itsolver_csr.c</a>	
Iterative solvers for <a href="#">dCSRmat</a> matrices	251
<a href="#">itsolver_mf.c</a>	
Iterative solvers using matrix-free spmv operations	257
<a href="#">itsolver_str.c</a>	
Iterative solvers for <a href="#">dSTRmat</a> matrices	260
<a href="#">lu.c</a>	
LU decomposition and direct solver for small dense matrices	264
<b>malloc.c.h</b>	??
<a href="#">memory.c</a>	
Memory allocation and deallocation subroutines	266
<a href="#">message.c</a>	
Output some useful messages	270
<a href="#">mgcycle.c</a>	
Abstract multigrid cycle – non-recursive version	273
<a href="#">mgrecur.c</a>	
Abstract multigrid cycle – recursive version	274
<b>nedmalloc.h</b>	??
<a href="#">ordering.c</a>	
Subroutines for ordering, merging, removing duplicated integers	275
<a href="#">parameters.c</a>	
Initialize, set, or print input data and parameters	281
<a href="#">pbcgs.c</a>	
Krylov subspace methods – Preconditioned BiCGstab	290
<a href="#">pbcgs_mf.c</a>	
Krylov subspace methods – Preconditioned BiCGstab (matrix free)	294
<a href="#">pcg.c</a>	
Krylov subspace methods – Preconditioned conjugate gradient	296
<a href="#">pcg_mf.c</a>	
Krylov subspace methods – Preconditioned conjugate gradient (matrix free)	302
<a href="#">pgcg.c</a>	
Krylov subspace methods – Preconditioned Generalized CG	304
<a href="#">pgcg_mf.c</a>	
Krylov subspace methods – Preconditioned Generalized CG (matrix free)	305
<a href="#">pgcr.c</a>	
Krylov subspace methods – Preconditioned GCR	306
<a href="#">pgmres.c</a>	
Krylov subspace methods – Right-preconditioned GMRes	309
<a href="#">pgmres_mf.c</a>	
Krylov subspace methods – Preconditioned GMRes (matrix free)	312
<a href="#">pminres.c</a>	
Krylov subspace methods – Preconditioned minimal residual	313
<a href="#">pminres_mf.c</a>	
Krylov subspace methods – Preconditioned minimal residual (matrix free)	317
<a href="#">precond_bcsr.c</a>	
Preconditioners for <a href="#">block_dCSRmat</a> matrices	319
<a href="#">precond_bsr.c</a>	
Preconditioners for <a href="#">dBSRmat</a> matrices	327
<a href="#">precond_csr.c</a>	
Preconditioners for <a href="#">dCSRmat</a> matrices	334

<a href="#">precond_str.c</a>	Preconditioners for <a href="#">dSTRmat</a> matrices . . . . .	340
<a href="#">pvfgmres.c</a>	Krylov subspace methods – Preconditioned variable-restarting flexible GMRes . . . . .	345
<a href="#">pvfgmres_mf.c</a>	Krylov subspace methods – Preconditioned variable-restarting flexible GMRes (matrix free) . . . . .	347
<a href="#">pvgmres.c</a>	Krylov subspace methods – Preconditioned variable-restart GMRes . . . . .	349
<a href="#">pvgmres_mf.c</a>	Krylov subspace methods – Preconditioned variable-restarting GMRes (matrix free) . . . . .	353
<a href="#">quadrature.c</a>	Quadrature rules . . . . .	354
<a href="#">rap.c</a>	Tripple-matrix multiplication $R \cdot A \cdot P$ . . . . .	355
<a href="#">schwarz_setup.c</a>	Setup phase for the Schwarz methods . . . . .	356
<a href="#">smat.c</a>	Simple operations for <i>small</i> dense matrices in row-major format . . . . .	359
<a href="#">smoother_bsr.c</a>	Smoothers for <a href="#">dBSRmat</a> matrices . . . . .	366
<a href="#">smoother_csr.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices . . . . .	376
<a href="#">smoother_csr_cr.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices using compatible relaxation . . . . .	382
<a href="#">smoother_csr_poly.c</a>	Smoothers for <a href="#">dCSRmat</a> matrices using poly. approx. to $A^{-1}$ . . . . .	383
<a href="#">smoother_str.c</a>	Smoothers for <a href="#">dSTRmat</a> matrices . . . . .	384
<a href="#">sparse_block.c</a>	Sparse matrix block operations . . . . .	393
<a href="#">sparse_bsr.c</a>	Sparse matrix operations for <a href="#">dBSRmat</a> matrices . . . . .	397
<a href="#">sparse_coo.c</a>	Sparse matrix operations for <a href="#">dCOOmat</a> matrices . . . . .	405
<a href="#">sparse_csr.c</a>	Sparse matrix operations for <a href="#">dCSRmat</a> matrices . . . . .	407
<a href="#">sparse_csrl.c</a>	Sparse matrix operations for <a href="#">dCSRLmat</a> matrices . . . . .	421
<a href="#">sparse_str.c</a>	Sparse matrix operations for <a href="#">dSTRmat</a> matrices . . . . .	422
<a href="#">sparse_util.c</a>	Routines for sparse matrix operations . . . . .	426
<a href="#">spbcgs.c</a>	Krylov subspace methods – Preconditioned BiCGstab with safety net . . . . .	433
<a href="#">spcg.c</a>	Krylov subspace methods – Preconditioned conjugate gradient with safety net . . . . .	438
<a href="#">spgmres.c</a>	Krylov subspace methods – Preconditioned GMRes with safety net . . . . .	442
<a href="#">spminres.c</a>	Krylov subspace methods – Preconditioned minimal residual with safety net . . . . .	446
<a href="#">spvgmres.c</a>	Krylov subspace methods – Preconditioned variable-restart GMRes with safety net . . . . .	450
<a href="#">threads.c</a>	Get and set number of threads and assign work load for each thread . . . . .	453

---

<a href="#">timing.c</a>	Timing subroutines . . . . .	455
<a href="#">vec.c</a>	Simple operations for vectors . . . . .	456
<a href="#">wrapper.c</a>	Wrappers for accessing functions by advanced users . . . . .	464





## Chapter 9

# Data Structure Documentation

### 9.1 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*
- [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*

### 9.1.1 Detailed Description

Data for AMG solvers.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 687 of file fasp.h.

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

- [fasp.h](#)

## 9.2 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 <= 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 dactorization from UMFPACK*
- [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*

### 9.2.1 Detailed Description

Data for multigrid levels. (BSR format)

#### Note

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

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

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

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

### 9.3 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 presmootherers*
- [SHORT postsmooth\\_iter](#)  
*number of postsmootherers*
- [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 amli\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [REAL \\* amli\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [SHORT nl\\_amli\\_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 ILUt*
- [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*

### 9.3.1 Detailed Description

Parameters for AMG solver.

#### Note

This is needed for the AMG solver/preconditioner.

Definition at line 548 of file fasp.h.

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

- [fasp.h](#)

## 9.4 block\_BSR Struct Reference

Block REAL matrix format for reservoir simulation.

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

### Data Fields

- [dBSRmat ResRes](#)  
*reservoir-reservoir block*
- [dCSRmat ResWel](#)  
*reservoir-well block*
- [dCSRmat WelRes](#)  
*well-reservoir block*
- [dCSRmat WelWel](#)  
*well-well block*

### 9.4.1 Detailed Description

Block REAL matrix format for reservoir simulation.

Definition at line 172 of file fasp\_block.h.

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

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

## 9.5 block\_dCSRmat 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]*

### 9.5.1 Detailed Description

Block REAL CSR matrix format.

#### Note

The starting index of A is 0.

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

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

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

## 9.6 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]*

### 9.6.1 Detailed Description

Block REAL vector structure.

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

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

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

## 9.7 block\_iCSRmat 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]*

### 9.7.1 Detailed Description

Block INT CSR matrix format.

#### Note

The starting index of A is 0.

Definition at line 103 of file fasp\_block.h.

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

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

## 9.8 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]*

### 9.8.1 Detailed Description

Block INT vector structure.

#### Note

The starting index of A is 0.

Definition at line 136 of file fasp\_block.h.

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

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



## 9.9 block\_Reservoir Struct Reference

Block REAL matrix format for reservoir simulation.

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

### Data Fields

- [dSTRmat ResRes](#)  
*reservoir-reservoir block*
- [dCSRmat ResWel](#)  
*reservoir-well block*
- [dCSRmat WelRes](#)  
*well-reservoir block*
- [dCSRmat WelWel](#)  
*well-well block*

### 9.9.1 Detailed Description

Block REAL matrix format for reservoir simulation.

Definition at line 151 of file fasp\_block.h.

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

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

## 9.10 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](#)

### 9.10.1 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 44 of file fasp\_block.h.

### 9.10.2 Field Documentation

#### 9.10.2.1 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 74 of file fasp\_block.h.

#### 9.10.2.2 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 67 of file fasp\_block.h.

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

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

## 9.11 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*

### 9.11.1 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 202 of file fasp.h.

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

- [fasp.h](#)

## 9.12 dCSRLmat Struct Reference

Sparse matrix of REAL type in CSRL format.

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

### Data Fields

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

### 9.12.1 Detailed Description

Sparse matrix of REAL type in CSRL format.

Definition at line 258 of file fasp.h.

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

- [fasp.h](#)

## 9.13 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*

### 9.13.1 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 141 of file fasp.h.

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

- [fasp.h](#)

## 9.14 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*

### 9.14.1 Detailed Description

Dense matrix of REAL type.

A dense REAL matrix

Definition at line 101 of file fasp.h.

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

- [fasp.h](#)

## 9.15 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])*

### 9.15.1 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 297 of file `fasp.h`.

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

- [fasp.h](#)

## 9.16 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*

### 9.16.1 Detailed Description

Vector with `n` entries of REAL type.

Definition at line 335 of file `fasp.h`.

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

- [fasp.h](#)

## 9.17 grid2d Struct Reference

Two dimensional grid data structure.

```
#include <fasp.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](#)

### 9.17.1 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 1089 of file [fasp.h](#).

### 9.17.2 Field Documentation

#### 9.17.2.1 [INT\(\\* e\)\[2\]](#)

Vertices of edges

Definition at line 1092 of file [fasp.h](#).

#### 9.17.2.2 [INT edges](#)

Number of edges

Definition at line 1103 of file [fasp.h](#).

#### 9.17.2.3 [INT\\* edir](#)

Boundary flags (0 <=> interior edge)

Definition at line 1096 of file [fasp.h](#).

#### 9.17.2.4 [INT\\* efather](#)

Father edge or triangle

Definition at line 1099 of file [fasp.h](#).

#### 9.17.2.5 [REAL\(\\* p\)\[2\]](#)

Coordinates of vertices

Definition at line 1091 of file [fasp.h](#).

#### 9.17.2.6 `INT* pdir`

Boundary flags (0 <=> interior point)

Definition at line 1095 of file fasp.h.

#### 9.17.2.7 `INT* pfather`

Father point or edge

Definition at line 1098 of file fasp.h.

#### 9.17.2.8 `INT(* s)[3]`

Edges of triangles

Definition at line 1094 of file fasp.h.

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

Vertices of triangles

Definition at line 1093 of file fasp.h.

#### 9.17.2.10 `INT* tfather`

Father triangle

Definition at line 1100 of file fasp.h.

#### 9.17.2.11 `INT triangles`

Number of triangles

Definition at line 1104 of file fasp.h.

#### 9.17.2.12 `INT vertices`

Number of grid points

Definition at line 1102 of file fasp.h.

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

- [fasp.h](#)

## 9.18 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*

### 9.18.1 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 232 of file fasp.h.

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

- [fasp.h](#)

## 9.19 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*

### 9.19.1 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 171 of file fasp.h.

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

- [fasp.h](#)

## 9.20 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*

### 9.20.1 Detailed Description

Dense matrix of INT type.

A dense INT matrix

Definition at line 120 of file fasp.h.

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

- [fasp.h](#)

## 9.21 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*

### 9.21.1 Detailed Description

Data for ILU setup.

Definition at line 393 of file fasp.h.

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

- [fasp.h](#)

## 9.22 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  $\text{permtol} * |a(i,j)| > |a(i,i)|$*

### 9.22.1 Detailed Description

Parameters for ILU.

Definition at line 367 of file fasp.h.

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

- [fasp.h](#)

## 9.23 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](#)

### 9.23.1 Detailed Description

Input parameters.

Input parameters, reading from disk file

Definition at line 991 of file fasp.h.

### 9.23.2 Field Documentation

#### 9.23.2.1 SHORT AMG\_aggregation\_type

aggregation type

Definition at line 1045 of file fasp.h.

#### 9.23.2.2 INT AMG\_aggressive\_level

number of levels use aggressive coarsening

Definition at line 1050 of file fasp.h.

#### 9.23.2.3 INT AMG\_aggressive\_path

number of paths used to determine strongly coupled C-set

Definition at line 1051 of file fasp.h.

#### 9.23.2.4 SHORT AMG\_amli\_degree

degree of the polynomial used by AMLI cycle

Definition at line 1039 of file fasp.h.

**9.23.2.5 INT AMG\_coarse\_dof**

max number of coarsest level DOF

Definition at line 1033 of file fasp.h.

**9.23.2.6 SHORT AMG\_coarse\_scaling**

switch of scaling of the coarse grid correction

Definition at line 1038 of file fasp.h.

**9.23.2.7 SHORT AMG\_coarse\_solver**

coarse solver type

Definition at line 1037 of file fasp.h.

**9.23.2.8 SHORT AMG\_coarsening\_type**

coarsening type

Definition at line 1044 of file fasp.h.

**9.23.2.9 SHORT AMG\_cycle\_type**

type of cycle

Definition at line 1026 of file fasp.h.

**9.23.2.10 SHORT AMG\_ILU\_levels**

how many levels use ILU smoother

Definition at line 1036 of file fasp.h.

**9.23.2.11 SHORT AMG\_interpolation\_type**

interpolation type

Definition at line 1046 of file fasp.h.

**9.23.2.12 SHORT AMG\_levels**

maximal number of levels

Definition at line 1025 of file fasp.h.

**9.23.2.13 INT AMG\_max\_aggregation**

max size of each aggregate

Definition at line 1057 of file fasp.h.

**9.23.2.14 REAL AMG\_max\_row\_sum**

maximal row sum

Definition at line 1049 of file fasp.h.

**9.23.2.15 INT AMG\_maxit**

number of iterations for AMG used as preconditioner

Definition at line 1035 of file fasp.h.

**9.23.2.16 SHORT AMG\_nl\_amli\_krylov\_type**

type of Krylov method used by nonlinear AMLI cycle

Definition at line 1040 of file fasp.h.

**9.23.2.17 INT AMG\_pair\_number**

number of pairs in matching algorithm

Definition at line 1052 of file fasp.h.

**9.23.2.18 SHORT AMG\_polynomial\_degree**

degree of the polynomial smoother

Definition at line 1030 of file fasp.h.

**9.23.2.19 SHORT AMG\_postsmooth\_iter**

number of postsmoothing

Definition at line 1032 of file fasp.h.

**9.23.2.20 SHORT AMG\_presmooth\_iter**

number of presmoothing

Definition at line 1031 of file fasp.h.

**9.23.2.21 REAL AMG\_quality\_bound**

threshold for pair wise aggregation

Definition at line 1053 of file fasp.h.

**9.23.2.22 REAL AMG\_relaxation**

over-relaxation parameter for SOR

Definition at line 1029 of file fasp.h.

**9.23.2.23 INT AMG\_Schwarz\_levels**

number of levels use Schwarz smoother

Definition at line 1041 of file fasp.h.

**9.23.2.24 SHORT AMG\_smooth\_filter**

use filter for smoothing the tentative prolongation or not

Definition at line 1059 of file fasp.h.

**9.23.2.25 SHORT AMG\_smooth\_order**

order for smoothers

Definition at line 1028 of file fasp.h.

**9.23.2.26 SHORT AMG\_smoother**

type of smoother

Definition at line 1027 of file fasp.h.

**9.23.2.27 REAL AMG\_strong\_coupled**

strong coupled threshold for aggregate

Definition at line 1056 of file fasp.h.

**9.23.2.28 REAL AMG\_strong\_threshold**

strong threshold for coarsening

Definition at line 1047 of file fasp.h.

**9.23.2.29 REAL AMG\_tentative\_smooth**

relaxation factor for smoothing the tentative prolongation

Definition at line 1058 of file fasp.h.

**9.23.2.30 REAL AMG\_tol**

tolerance for AMG if used as preconditioner

Definition at line 1034 of file fasp.h.

**9.23.2.31 REAL AMG\_truncation\_threshold**

truncation factor for interpolation

Definition at line 1048 of file fasp.h.



**9.23.2.32 SHORT AMG\_type**

Type of AMG

Definition at line 1024 of file fasp.h.

**9.23.2.33 REAL ILU\_droptol**

drop tolerance

Definition at line 1013 of file fasp.h.

**9.23.2.34 INT ILU\_lfil**

level of fill-in

Definition at line 1012 of file fasp.h.

**9.23.2.35 REAL ILU\_permtol**

permutation tolerance

Definition at line 1015 of file fasp.h.

**9.23.2.36 REAL ILU\_relax**

scaling factor: add the sum of dropped entries to diagonal

Definition at line 1014 of file fasp.h.

**9.23.2.37 SHORT ILU\_type**

ILU type for decomposition

Definition at line 1011 of file fasp.h.

**9.23.2.38 char inifile[256]**

ini file name

Definition at line 998 of file fasp.h.

**9.23.2.39 INT itsolver\_maxit**

maximal number of iterations for iterative solvers

Definition at line 1007 of file fasp.h.

**9.23.2.40 REAL itsolver\_tol**

tolerance for iterative linear solver

Definition at line 1006 of file fasp.h.

**9.23.2.41 SHORT output\_type**

type of output stream

Definition at line 995 of file fasp.h.

**9.23.2.42 SHORT precondition\_type**

type of preconditioner for iterative solvers

Definition at line 1004 of file fasp.h.

**9.23.2.43 SHORT print\_level**

print level

Definition at line 994 of file fasp.h.

**9.23.2.44 INT problem\_num**

problem number to solve

Definition at line 1000 of file fasp.h.

**9.23.2.45 INT restart**

restart number used in GMRES

Definition at line 1008 of file fasp.h.

**9.23.2.46 INT Schwarz\_blksolver**

type of Schwarz block solver

Definition at line 1021 of file fasp.h.

**9.23.2.47 INT Schwarz\_maxlvl**

maximal levels

Definition at line 1019 of file fasp.h.

**9.23.2.48 INT Schwarz\_mmsize**

maximal block size

Definition at line 1018 of file fasp.h.

**9.23.2.49 INT Schwarz\_type**

type of Schwarz method

Definition at line 1020 of file fasp.h.

#### 9.23.2.50 **SHORT** solver\_type

type of iterative solvers

Definition at line 1003 of file fasp.h.

#### 9.23.2.51 **SHORT** stop\_type

type of stopping criteria for iterative solvers

Definition at line 1005 of file fasp.h.

#### 9.23.2.52 **char** workdir[256]

working directory for data files

Definition at line 999 of file fasp.h.

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

- [fasp.h](#)

## 9.24 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](#)

### 9.24.1 Detailed Description

Parameters passed to iterative solvers.

Definition at line 1067 of file fasp.h.

### 9.24.2 Field Documentation

#### 9.24.2.1 **SHORT** itsolver\_type

solver type: see message.h

Definition at line 1069 of file fasp.h.

#### 9.24.2.2 INT maxit

max number of iterations

Definition at line 1072 of file fasp.h.

#### 9.24.2.3 SHORT precond\_type

preconditioner type: see message.h

Definition at line 1070 of file fasp.h.

#### 9.24.2.4 SHORT print\_level

print level: 0–10

Definition at line 1075 of file fasp.h.

#### 9.24.2.5 INT restart

number of steps for restarting: for GMRES etc

Definition at line 1074 of file fasp.h.

#### 9.24.2.6 SHORT stop\_type

stopping criteria type

Definition at line 1071 of file fasp.h.

#### 9.24.2.7 REAL tol

convergence tolerance

Definition at line 1073 of file fasp.h.

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

- [fasp.h](#)

## 9.25 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*

### 9.25.1 Detailed Description

Vector with n entries of INT type.

Definition at line 349 of file fasp.h.

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

- [fasp.h](#)

## 9.26 Link Struct Reference

Struct for Links.

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

### Data Fields

- [INT prev](#)  
*previous node in the linklist*
- [INT next](#)  
*next node in the linklist*

### 9.26.1 Detailed Description

Struct for Links.

Definition at line 1116 of file fasp.h.

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

- [fasp.h](#)

## 9.27 linked\_list Struct Reference

A linked list node.

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

### Data Fields

- [INT data](#)  
*data*
- [INT head](#)  
*starting of the list*
- [INT tail](#)  
*ending of the list*
- struct [linked\\_list](#) \* [next\\_node](#)  
*next node*
- struct [linked\\_list](#) \* [prev\\_node](#)  
*previous node*

### 9.27.1 Detailed Description

A linked list node.

#### Note

This definition is adapted from hypre 2.0.

Definition at line 1133 of file fasp.h.

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

- [fasp.h](#)

## 9.28 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**

### 9.28.1 Detailed Description

Definition at line 69 of file dlmalloc.h.

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

- dlmalloc.h
- malloc.c.h

## 9.29 malloc\_chunk Struct Reference

### Data Fields

- size\_t **prev\_foot**
- size\_t **head**
- struct [malloc\\_chunk](#) \* **fd**
- struct [malloc\\_chunk](#) \* **bk**

### 9.29.1 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

## 9.30 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**

### 9.30.1 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

## 9.31 malloc\_segment Struct Reference

### Data Fields

- char \* **base**
- size\_t **size**
- struct [malloc\\_segment](#) \* **next**
- flag\_t **sflags**

### 9.31.1 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

## 9.32 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**

### 9.32.1 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`

## 9.33 `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**

### 9.33.1 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`



## 9.34 Mumps\_data Struct Reference

Parameters for MUMPS interface.

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

### Data Fields

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

#### 9.34.1 Detailed Description

Parameters for MUMPS interface.

Added on 10/10/2014

Definition at line 452 of file fasp.h.

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

- [fasp.h](#)

## 9.35 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](#) )(void \*, [REAL](#) \*, [REAL](#) \*)  
*action for MxV, void function pointer*

#### 9.35.1 Detailed Description

Matrix-vector multiplication, replace the actual matrix.

Definition at line 975 of file fasp.h.

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

- [fasp.h](#)

## 9.36 nedmallinfo Struct Reference

### Data Fields

- size\_t [arena](#)

- `size_t ordblks`
- `size_t smblks`
- `size_t hblks`
- `size_t hblkhd`
- `size_t usmblks`
- `size_t fsmblks`
- `size_t uordblks`
- `size_t fordblks`
- `size_t keepcost`

### 9.36.1 Detailed Description

Definition at line 168 of file `nedmalloc.h`.

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

- `nedmalloc.h`

## 9.37 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*

### 9.37.1 Detailed Description

Preconditioner data and action.

#### Note

This is the preconditioner structure for preconditioned iterative methods.

Definition at line 961 of file `fasp.h`.

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

- `fasp.h`

## 9.38 precondition\_block\_data Struct Reference

Data passed to the preconditioner for block preconditioning for `block_dCSRmat` format.

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

## Data Fields

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

### 9.38.1 Detailed Description

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

This is needed for the block preconditioner.

Definition at line 499 of file [fasp\\_block.h](#).

### 9.38.2 Field Documentation

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

data for each diagonal block

Definition at line 506 of file [fasp\\_block.h](#).

#### 9.38.2.2 [block\\_dCSRmat](#)\* [Abcsr](#)

problem data, the blocks

Definition at line 504 of file [fasp\\_block.h](#).

#### 9.38.2.3 [AMG\\_param](#)\* [amgparam](#)

parameters for AMG

Definition at line 518 of file [fasp\\_block.h](#).

#### 9.38.2.4 void\*\* [LU\\_diag](#)

LU decomposition for the diagonal blocks (for UMFPack)

Definition at line 514 of file [fasp\\_block.h](#).

#### 9.38.2.5 [AMG\\_data](#)\*\* [mgl](#)

AMG data for the diagonal blocks

Definition at line 517 of file [fasp\\_block.h](#).

### 9.38.2.6 `dvector r`

temp work space

Definition at line 508 of file `fasp_block.h`.

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

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

## 9.39 `precond_block_reservoir_data` Struct Reference

Data passed to the preconditioner for reservoir simulation problems.

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

### Data Fields

- [block\\_Reservoir](#) \* [A](#)  
*problem data in [block\\_Reservoir](#) format*
- [block\\_dCSRmat](#) \* [Abcsr](#)  
*problem data in [block\\_dCSRmat](#) format*
- [dCSRmat](#) \* [Acsr](#)  
*problem data in CSR format*
- [INT](#) [ILU\\_ifil](#)  
*level of fill-in for structured ILU(k)*
- [dSTRmat](#) \* [LU](#)  
*LU matrix for Reservoir-Reservoir block in STR format.*
- [ILU\\_data](#) \* [LUcsr](#)  
*LU matrix for Reservoir-Reservoir block in CSR format.*
- [AMG\\_data](#) \* [mgl\\_data](#)  
*AMG data for presure-presure block.*
- [SHORT](#) [print\\_level](#)  
*print level in AMG preconditioner*
- [INT](#) [maxit\\_AMG](#)  
*max number of iterations of AMG preconditioner*
- [SHORT](#) [max\\_levels](#)  
*max number of AMG levels*
- [REAL](#) [amg\\_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 coarse grid correction*
- [INT maxit](#)  
*max number of iterations*
- [INT restart](#)  
*number of iterations for restart*
- [REAL tol](#)  
*tolerance for convergence*
- [REAL \\* invS](#)  
*inverse of the Schur complement  $(-I - A_{wr} * A_{rr}^{-1} * A_{rw})^{-1}$ ,  $A_{rr}$  may be replaced by LU*
- [dvector \\* DPSinvDSS](#)  
 *$Diag(PS) * inv(Diag(SS))$*
- [SHORT scaled](#)
- [ivector \\* perf\\_idx](#)
- [dSTRmat \\* RR](#)
- [dCSRmat \\* WW](#)
- [dCSRmat \\* PP](#)
- [dSTRmat \\* SS](#)
- [precond\\_diagstr \\* diag](#)
- [dvector \\* diaginv](#)
- [ivector \\* pivot](#)
- [dvector \\* diaginvS](#)
- [ivector \\* pivotS](#)
- [ivector \\* order](#)
- [dvector r](#)
- [REAL \\* w](#)

### 9.39.1 Detailed Description

Data passed to the preconditioner for reservoir simulation problems.

#### Note

This is only needed for the Black Oil model with wells

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

### 9.39.2 Field Documentation

#### 9.39.2.1 precondition\_diagstr\* diag

the diagonal inverse for diagonal scaling

Definition at line 481 of file fasp\_block.h.

**9.39.2.2 dvector\* diaginV**

the inverse of the diagonals for GS/block GS smoother (whole reservoir matrix)

Definition at line 482 of file fasp\_block.h.

**9.39.2.3 dvector\* diaginVS**

the inverse of the diagonals for GS/block GS smoother (saturation block)

Definition at line 484 of file fasp\_block.h.

**9.39.2.4 ivector\* order**

order for smoothing

Definition at line 486 of file fasp\_block.h.

**9.39.2.5 ivector\* perf\_idx**

variable index for perf

Definition at line 474 of file fasp\_block.h.

**9.39.2.6 ivector\* pivot**

the pivot for the GS/block GS smoother (whole reservoir matrix)

Definition at line 483 of file fasp\_block.h.

**9.39.2.7 ivector\* pivotS**

the pivot for the GS/block GS smoother (saturation block)

Definition at line 485 of file fasp\_block.h.

**9.39.2.8 dCSRmat\* PP**

pressure block after diagonal scaling

Definition at line 478 of file fasp\_block.h.

**9.39.2.9 dvector r**

temporary dvector used to store and restore the residual

Definition at line 489 of file fasp\_block.h.

**9.39.2.10 dSTRmat\* RR**

Diagonal scaled reservoir block

Definition at line 476 of file fasp\_block.h.

**9.39.2.11 `SHORT` scaled**

whether the matrix is scaled

Definition at line 473 of file `fasp_block.h`.

**9.39.2.12 `dSTRmat*` SS**

saturation block after diagonal scaling

Definition at line 479 of file `fasp_block.h`.

**9.39.2.13 `REAL*` w**

temporary work space for other usage

Definition at line 490 of file `fasp_block.h`.

**9.39.2.14 `dCSRmat*` WW**

Argumented well block

Definition at line 477 of file `fasp_block.h`.

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

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

**9.40 `precond_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*

### 9.40.1 Detailed Description

Data passed to the preconditioners.

Definition at line 757 of file fasp.h.

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

- [fasp.h](#)



## 9.41 precondition\_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 amli\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [REAL \\* amli\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [REAL tentative\\_smooth](#)  
*smooth factor for smoothing the tentative prolongation*
- [SHORT nl\\_amli\\_krylov\\_type](#)  
*type of krylov method used by Nonlinear AMLI cycle*
- [AMG\\_data\\_bsr \\* mgl\\_data](#)  
*AMG preconditioner data.*
- [AMG\\_data \\* pres\\_mgl\\_data](#)

- [AMG preconditioner data for pressure block.](#)
- [ILU\\_data \\* LU](#)  
*ILU preconditioner data (needed for CPR type preconditioner)*
- [dBSRmat \\* A](#)  
*Matrix data.*
- [dCSRmat \\* A\\_nk](#)  
*Matrix data for near kernal.*
- [dCSRmat \\* P\\_nk](#)  
*Prolongation for near kernal.*
- [dCSRmat \\* R\\_nk](#)  
*Resriction for near kernal.*
- [dvector r](#)  
*temporary dvector used to store and restore the residual*
- [REAL \\* w](#)  
*temporary work space for other usage*

### 9.41.1 Detailed Description

Data passed to the preconditioners.

#### Note

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

Definition at line 308 of file fasp\_block.h.

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

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

## 9.42 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](#) \* [diaginv](#)  
*the inverse of the diagonals for GS/block GS smoother (whole reservoir matrix)*
- [ivector](#) \* [pivot](#)  
*the pivot for the GS/block GS smoother (whole reservoir matrix)*
- [dvector](#) \* [diaginvS](#)  
*the inverse of the diagonals for GS/block GS smoother (saturation block)*
- [ivector](#) \* [pivotS](#)  
*the pivot for the GS/block GS smoother (saturation block)*
- [ivector](#) \* [order](#)  
*order for smoothing*
- [ivector](#) \* [neigh](#)  
*array to store neighbor information*
- [dvector](#) [r](#)  
*temporary dvector used to store and restore the residual*
- [REAL](#) \* [w](#)  
*temporary work space for other usage*

### 9.42.1 Detailed Description

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

Definition at line 853 of file fasp.h.

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

- [fasp.h](#)

## 9.43 `precond_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*

### 9.43.1 Detailed Description

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

#### Note

This is needed for the diagonal preconditioner.

Definition at line 290 of file fasp\_block.h.

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

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

## 9.44 `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*

### 9.44.1 Detailed Description

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

#### Note

This is needed for the diagonal preconditioner.

Definition at line 945 of file fasp.h.

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

- [fasp.h](#)

## 9.45 precondition\_FASP\_blkoi\_data Struct Reference

Data passed to the preconditioner for preconditioning reservoir simulation problems.

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

### Data Fields

- [block\\_BSR](#) \* A  
*Part 1: Basic data.*
- [SHORT](#) scaled  
*Part 2: Data for CPR-like preconditioner for reservoir block.*
- [dvector](#) \* [diaginv\\_noscale](#)
- [dBSRmat](#) \* RR
- [ivector](#) \* [neigh](#)
- [ivector](#) \* [order](#)
- [dBSRmat](#) \* SS
- [dvector](#) \* [diaginv\\_S](#)
- [ivector](#) \* [pivot\\_S](#)
- [dCSRmat](#) \* PP
- [AMG\\_data](#) \* [mgl\\_data](#)
- [SHORT](#) [print\\_level](#)  
*print level in AMG preconditioner*
- [INT](#) [maxit\\_AMG](#)  
*max number of iterations of AMG preconditioner*
- [SHORT](#) [max\\_levels](#)  
*max number of AMG levels*
- [REAL](#) [amg\\_tol](#)  
*tolerance for AMG preconditioner*
- [SHORT](#) [cycle\\_type](#)  
*AMG cycle type.*
- [SHORT](#) [smoother](#)  
*AMG smoother type.*
- [SHORT](#) [smooth\\_order](#)  
*AMG smoothing order.*

- [SHORT presmooth\\_iter](#)  
*number of presmoothing*
- [SHORT postsmooth\\_iter](#)  
*number of postsmoothing*
- [SHORT coarsening\\_type](#)  
*coarsening type*
- [INT coarse\\_dof](#)  
*coarset dof*
- [SHORT coarse\\_solver](#)  
*coarse level solver type*
- [REAL relaxation](#)  
*relaxation parameter for SOR smoother*
- [SHORT coarse\\_scaling](#)  
*switch of scaling of coarse grid correction*
- [SHORT amli\\_degree](#)  
*degree of the polynomial used by AMLI cycle*
- [REAL \\* amli\\_coef](#)  
*coefficients of the polynomial used by AMLI cycle*
- [REAL tentative\\_smooth](#)  
*relaxation parameter for smoothing the tentative prolongation*
- [dvector \\* diaginv](#)
- [ivector \\* pivot](#)
- [ILU\\_data \\* LU](#)  
*data of ILU for reservoir block*
- [ivector \\* perf\\_idx](#)
- [ivector \\* perf\\_neigh](#)
- [dCSRmat \\* WW](#)
- [void \\* Numeric](#)  
*data for direct solver for argumented well block*
- [REAL \\* invS](#)  
*inverse of the schur complement  $(-I - Awr*Arr^{-1}*Arw)^{-1}$ , Arr may be replaced by LU*
- [INT maxit](#)
- [INT restart](#)
- [REAL tol](#)
- [dvector r](#)
- [REAL \\* w](#)

### 9.45.1 Detailed Description

Data passed to the preconditioner for preconditioning reservoir simulation problems.

#### Note

This is only needed for the Black Oil model with wells

Definition at line 528 of file fasp\_block.h.

## 9.45.2 Field Documentation

### 9.45.2.1 **block\_BSR\*** A

Part 1: Basic data.

whole jacobian system in block\_BSRmat

Definition at line 533 of file fasp\_block.h.

### 9.45.2.2 **dvector\*** diaginv

inverse of the diagonal blocks of reservoir block

Definition at line 608 of file fasp\_block.h.

### 9.45.2.3 **dvector\*** diaginv\_noscale

inverse of diagonal blocks for diagonal scaling

Definition at line 540 of file fasp\_block.h.

### 9.45.2.4 **dvector\*** diaginv\_S

inverse of the diagonal blocks of saturation block

Definition at line 549 of file fasp\_block.h.

### 9.45.2.5 **INT** maxit

max number of iterations

Definition at line 626 of file fasp\_block.h.

### 9.45.2.6 **AMG\_data\*** mgl\_data

AMG data for presure-presure block

Definition at line 554 of file fasp\_block.h.

### 9.45.2.7 **ivector\*** neigh

neighbor information of the reservoir block

Definition at line 544 of file fasp\_block.h.

### 9.45.2.8 **ivector\*** order

ordering of the reservoir block

Definition at line 545 of file fasp\_block.h.

**9.45.2.9    ivector\* perf\_idx**

index of blocks which have perforation

Definition at line 615 of file fasp\_block.h.

**9.45.2.10    ivector\* perf\_neigh**

index of blocks which are neighbors of perforations (include perforations)

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

**9.45.2.11    ivector\* pivot**

pivot for the GS smoothers for the reservoir matrix

Definition at line 609 of file fasp\_block.h.

**9.45.2.12    ivector\* pivot\_S**

pivoting for the GS smoothers for saturation block

Definition at line 550 of file fasp\_block.h.

**9.45.2.13    dCSRmat\* PP**

pressure block

Definition at line 553 of file fasp\_block.h.

**9.45.2.14    dvector r**

temporary dvector used to store and restore the residual

Definition at line 631 of file fasp\_block.h.

**9.45.2.15    INT restart**

number of iterations for restart

Definition at line 627 of file fasp\_block.h.

**9.45.2.16    dBSRmat\* RR**

reservoir block

Definition at line 541 of file fasp\_block.h.

**9.45.2.17    SHORT scaled**

Part 2: Data for CPR-like preconditioner for reservoir block.



scaled = 1 means the the following RR block is diagonal scaled

Definition at line 539 of file fasp\_block.h.

#### 9.45.2.18 dBSRmat\* SS

saturation block

Definition at line 548 of file fasp\_block.h.

#### 9.45.2.19 REAL tol

tolerance

Definition at line 628 of file fasp\_block.h.

#### 9.45.2.20 REAL\* w

temporary work space for other usage

Definition at line 632 of file fasp\_block.h.

#### 9.45.2.21 dCSRmat\* WW

Argumented well block

Definition at line 617 of file fasp\_block.h.

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

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

## 9.46 precondition\_sweeping\_data Struct Reference

Data passed to the preconditioner for sweeping preconditioning.

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

### Data Fields

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

### 9.46.1 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 645 of file fasp\_block.h.

### 9.46.2 Field Documentation

#### 9.46.2.1 `block_dCSRmat*` A

problem data, the sparse matrix

Definition at line 649 of file fasp\_block.h.

#### 9.46.2.2 `block_dCSRmat*` Ai

preconditioner data, the sparse matrix

Definition at line 650 of file fasp\_block.h.

#### 9.46.2.3 `dCSRmat*` local\_A

local stiffness matrix for each layer

Definition at line 652 of file fasp\_block.h.

#### 9.46.2.4 `ivector*` local\_index

local index for each layer

Definition at line 655 of file fasp\_block.h.

#### 9.46.2.5 `void**` local\_LU

local LU decomposition (for UMFPack)

Definition at line 653 of file fasp\_block.h.

## 9.46.2.6 INT NumLayers

number of layers

Definition at line 647 of file fasp\_block.h.

## 9.46.2.7 dvector r

temporary dvector used to store and restore the residual

Definition at line 658 of file fasp\_block.h.

## 9.46.2.8 REAL\* w

temporary work space for other usage

Definition at line 659 of file fasp\_block.h.

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

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

## 9.47 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*

### 9.47.1 Detailed Description

Data for Schwarz methods.

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

Definition at line 470 of file fasp.h.

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

- [fasp.h](#)

## 9.48 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*

### 9.48.1 Detailed Description

Parameters for Schwarz method.

Added on 05/14/2012

Definition at line 427 of file fasp.h.

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

- [fasp.h](#)



## Chapter 10

# File Documentation

### 10.1 amg.c File Reference

AMG method as an iterative solver (main file)

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

#### Functions

- void [fasp\\_solver\\_amg](#) (dCSRmat \*A, dvector \*b, dvector \*x, AMG\_param \*param)  
*Solve  $Ax = b$  by algebraic multigrid methods.*

#### 10.1.1 Detailed Description

AMG method as an iterative solver (main file)

#### 10.1.2 Function Documentation

10.1.2.1 void [fasp\\_solver\\_amg](#) ( dCSRmat \* A, 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 37 of file amg.c.

## 10.2 amg\_setup\_cr.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.*

### 10.2.1 Detailed Description

Brannick-Falgout compatible relaxation based AMG: SETUP phase.

**Note**

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

### 10.2.2 Function Documentation

#### 10.2.2.1 [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 38 of file amg\_setup\_cr.c.

## 10.3 amg\_setup\_rs.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.*

### 10.3.1 Detailed Description

Ruge-Stuben AMG: SETUP phase.

## Note

Ref 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.

### 10.3.2 Function Documentation

#### 10.3.2.1 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 47 of file amg\_setup\_rs.c.

## 10.4 amg\_setup\_sa.c File Reference

Smoothed aggregation AMG: SETUP phase.

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

## Functions

- [SHORT fasp\\_amg\\_setup\\_sa](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of smoothed aggregation AMG.*
- [SHORT fasp\\_amg\\_setup\\_sa\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of smoothed aggregation AMG (BSR format)*

### 10.4.1 Detailed Description

Smoothed aggregation AMG: SETUP phase.

## Note

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

### 10.4.2 Function Documentation

#### 10.4.2.1 **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 48 of file amg\_setup\_sa.c.

#### 10.4.2.2 **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 85 of file amg\_setup\_sa.c.

## 10.5 amg\_setup\_ua.c File Reference

Unsmoothed aggregation AMG: SETUP phase.

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

### Functions

- [SHORT fasp\\_amg\\_setup\\_ua](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of unsmoothed aggregation AMG.*
- [SHORT fasp\\_amg\\_setup\\_ua\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
*Set up phase of unsmoothed aggregation AMG (BSR format)*

### 10.5.1 Detailed Description

Unsmoothed aggregation AMG: SETUP phase.

#### Note

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

### 10.5.2 Function Documentation

#### 10.5.2.1 [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 38 of file amg\_setup\_ua.c.

10.5.2.2 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 69 of file amg\_setup\_ua.c.

## 10.6 amg\_solve.c File Reference

Algebraic multigrid iterations: SOLVE phase.

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

**Functions**

- [INT fasp\\_amg\\_solve](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*AMG – SOLVE phase.*
- [INT fasp\\_amg\\_solve\\_aml](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*AMLI – SOLVE phase.*
- [INT fasp\\_amg\\_solve\\_nl\\_aml](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*Nonlinear AMLI – SOLVE phase.*
- [void fasp\\_famg\\_solve](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*FMG – SOLVE phase.*

### 10.6.1 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!

## 10.6.2 Function Documentation

10.6.2.1 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 36 of file amg\_solve.c.

**10.6.2.2** `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 125 of file amg\_solve.c.

**10.6.2.3** `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 Xiazhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid", 2013.

Definition at line 209 of file amg\_solve.c.

10.6.2.4 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 281 of file amg\_solve.c.

## 10.7 amlirecur.c File Reference

Abstract AMLI multilevel iteration – recursive version.

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

### 10.7.1 Detailed Description

Abstract AMLI multilevel iteration – recursive version.

#### Note

AMLI and non-linear AMLI cycles

### 10.7.2 Function Documentation

10.7.2.1 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 679 of file amlirecur.c.

10.7.2.2 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.

Definition at line 44 of file amlirecur.c.

**10.7.2.3** 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 AMLI-cycle Multigrid", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Zheng Li on 11/10/2014: update direct solvers.

Definition at line 259 of file amlirecur.c.

**10.7.2.4** 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.

Definition at line 489 of file amlirecur.c.

## 10.8 array.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, [REAL](#) \*x, [REAL](#) \*y)  
*Copy an array to the other y=x.*
- void [fasp\\_iarray\\_cp](#) (const [INT](#) n, [INT](#) \*x, [INT](#) \*y)  
*Copy an array to the other y=x.*
- void [fasp\\_array\\_cp\\_nc3](#) ([REAL](#) \*x, [REAL](#) \*y)  
*Copy an array to the other y=x, the length is 3.*
- void [fasp\\_array\\_cp\\_nc5](#) ([REAL](#) \*x, [REAL](#) \*y)  
*Copy an array to the other y=x, the length is 5.*
- void [fasp\\_array\\_cp\\_nc7](#) ([REAL](#) \*x, [REAL](#) \*y)  
*Copy an array to the other y=x, the length is 7.*

### 10.8.1 Detailed Description

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

### 10.8.2 Function Documentation

#### 10.8.2.1 void fasp\_array\_cp ( const INT *n*, 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 165 of file array.c.

#### 10.8.2.2 void fasp\_array\_cp\_nc3 ( 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 205 of file array.c.

#### 10.8.2.3 void fasp\_array\_cp\_nc5 ( REAL \* *x*, REAL \* *y* )

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

**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 226 of file array.c.

**10.8.2.4 void fasp\_array\_cp\_nc7 ( 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 249 of file array.c.

**10.8.2.5 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 29 of file array.c.

**10.8.2.6 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 48 of file array.c.

**10.8.2.7 void fasp\_iallay\_cp ( const INT *n*, 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 185 of file array.c.

**10.8.2.8 void fasp\_iallay\_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 107 of file array.c.

## 10.9 blas\_array.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, `REAL` \*x, `REAL` \*y)  
 $y = a * x + y$
- void `fasp_blas_array_axpyz` (const `INT` n, const `REAL` a, `REAL` \*x, `REAL` \*y, `REAL` \*z)  
 $z = a * x + y$
- void `fasp_blas_array_axpby` (const `INT` n, const `REAL` a, `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.*

### 10.9.1 Detailed Description

BLAS1 operations for arrays.



## 10.9.2 Function Documentation

10.9.2.1 void fasp\_blas\_array\_ax ( const INT  $n$ , const REAL  $a$ , REAL \*  $x$  )

$x = a * x$

## Parameters

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

## Author

Chensong Zhang

## Date

07/01/209

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

## Note

x is reused to store the resulting array.

Definition at line 35 of file blas\_array.c.

10.9.2.2 void fasp\_blas\_array\_axpy ( const INT  $n$ , const REAL  $a$ , 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

07/01/209

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

## Note

y is reused to store the resulting array.

Definition at line 218 of file blas\_array.c.

10.9.2.3 void fasp\_blas\_array\_axpy ( const INT  $n$ , const REAL  $a$ , 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 87 of file blas\_array.c.

10.9.2.4 void fasp\_blas\_array\_axpyz ( const INT  $n$ , const REAL  $a$ , REAL \*  $x$ , 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 167 of file blas\_array.c.

10.9.2.5 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**

07/01/209

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

Definition at line 267 of file blas\_array.c.

**10.9.2.6 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 307 of file blas\_array.c.

**10.9.2.7 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

07/01/2009

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

Definition at line 347 of file blas\_array.c.

### 10.9.2.8 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

07/01/2009

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

Definition at line 388 of file blas\_array.c.

## 10.10 blas\_bcsr.c File Reference

BLAS2 operations for [block\\_dCSRmat](#) matrices.

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

## Functions

- void `fasp_blas_bdcscr_aAxy` (const `REAL` `alpha`, `block_dCSRmat` \*`A`, `REAL` \*`x`, `REAL` \*`y`)  
*Matrix-vector multiplication  $y = \alpha A * x + y$ .*
- void `fasp_blas_bdcscr_mxv` (`block_dCSRmat` \*`A`, `REAL` \*`x`, `REAL` \*`y`)  
*Matrix-vector multiplication  $y = A * x$ .*
- void `fasp_blas_bdbsr_aAxy` (const `REAL` `alpha`, `block_BSR` \*`A`, `REAL` \*`x`, `REAL` \*`y`)  
*Matrix-vector multiplication  $y = \alpha A * x + y$ .*
- void `fasp_blas_bdbsr_mxv` (`block_BSR` \*`A`, `REAL` \*`x`, `REAL` \*`y`)  
*Matrix-vector multiplication  $y = A * x$ .*

### 10.10.1 Detailed Description

BLAS2 operations for `block_dCSRmat` matrices.

### 10.10.2 Function Documentation

10.10.2.1 void `fasp_blas_bdbsr_aAxy` ( const `REAL` `alpha`, `block_BSR` \* `A`, `REAL` \* `x`, `REAL` \* `y` )

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

#### Parameters

<code>alpha</code>	REAL factor a
<code>A</code>	Pointer to <code>block_BSR</code> matrix A
<code>x</code>	Pointer to array x
<code>y</code>	Pointer to array y

#### Author

Xiaozhe Hu

#### Date

11/11/2010

Definition at line 288 of file `blas_bcsr.c`.

10.10.2.2 void `fasp_blas_bdbsr_mxv` ( `block_BSR` \* `A`, `REAL` \* `x`, `REAL` \* `y` )

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

#### Parameters

<code>A</code>	Pointer to <code>block_BSR</code> matrix A
<code>x</code>	Pointer to array x
<code>y</code>	Pointer to array y

#### Author

Xiaozhe Hu

## Date

11/11/2010

Definition at line 326 of file blas\_bcsr.c.

10.10.2.3 void fasp\_blas\_bdcsr\_aApy ( const REAL *alpha*, block\_dCSRmat \* *A*, 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="#">block_dCSRmat</a> matrix <i>A</i>
<i>x</i>	Pointer to array <i>x</i>
<i>y</i>	Pointer to array <i>y</i>

## Author

Xiaozhe Hu

## Date

06/04/2010

Definition at line 30 of file blas\_bcsr.c.

10.10.2.4 void fasp\_blas\_bdcsr\_mxv ( block\_dCSRmat \* *A*, REAL \* *x*, REAL \* *y* )

Matrix-vector multiplication  $y = Ax$ .

## Parameters

<i>A</i>	Pointer to <a href="#">block_dCSRmat</a> matrix <i>A</i>
<i>x</i>	Pointer to array <i>x</i>
<i>y</i>	Pointer to array <i>y</i>

## Author

Chensong Zhang

## Date

04/27/2013

Definition at line 155 of file blas\_bcsr.c.

## 10.11 blas\_bsr.c File Reference

BLAS2 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`, `dBSRmat *A`, `REAL *x`, `REAL *y`)  
*Compute  $y := \alpha A * x + y$ .*
- void `fasp_blas_dbsr_aAxy_agg` (const `REAL alpha`, `dBSRmat *A`, `REAL *x`, `REAL *y`)  
*Compute  $y := \alpha A * x + y$  where each small block matrix is an identity matrix.*
- void `fasp_blas_dbsr_m xv` (`dBSRmat *A`, `REAL *x`, `REAL *y`)  
*Compute  $y := A * x$ .*
- void `fasp_blas_dbsr_m xv_agg` (`dBSRmat *A`, `REAL *x`, `REAL *y`)  
*Compute  $y := A * x$ , where each small block matrices of A is an identity matrix.*
- void `fasp_blas_dbsr_m xm` (`dBSRmat *A`, `dBSRmat *B`, `dBSRmat *C`)  
*Sparse matrix multiplication  $C = A * B$ .*
- void `fasp_blas_dbsr_rap1` (`dBSRmat *R`, `dBSRmat *A`, `dBSRmat *P`, `dBSRmat *B`)  
*`dBSRmat` sparse matrix multiplication  $B = R * A * P$*
- void `fasp_blas_dbsr_rap` (`dBSRmat *R`, `dBSRmat *A`, `dBSRmat *P`, `dBSRmat *B`)  
*`dBSRmat` sparse matrix multiplication  $B = R * A * P$*
- void `fasp_blas_dbsr_rap_agg` (`dBSRmat *R`, `dBSRmat *A`, `dBSRmat *P`, `dBSRmat *B`)  
*`dBSRmat` sparse matrix multiplication  $B = R * A * P$ , where small block matrices in P and R are identity matrices!*

### 10.11.1 Detailed Description

BLAS2 operations for `dBSRmat` matrices.

### 10.11.2 Function Documentation

10.11.2.1 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 <code>dBSRmat</code> 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 59 of file blas\_bsr.c.

10.11.2.2 void fasp\_blas\_dbsr\_aApy ( const **REAL** *alpha*, **dBSRmat** \* *A*, **REAL** \* *x*, **REAL** \* *y* )

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

**Parameters**

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the <b>dBSRmat</b> 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 337 of file blas\_bsr.c.

10.11.2.3 void fasp\_blas\_dbsr\_aApy\_agg ( const **REAL** *alpha*, **dBSRmat** \* *A*, **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 <b>dBSRmat</b> 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 610 of file blas\_bsr.c.

10.11.2.4 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 30 of file blas\_bsr.c.

10.11.2.5 void fasp\_blas\_dbsr\_mxm ( [dBSRmat](#) \* *A*, [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 4591 of file blas\_bsr.c.

10.11.2.6 void fasp\_blas\_dbsr\_m xv ( [dBSRmat](#) \* *A*, 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 895 of file blas\_bsr.c.

10.11.2.7 void fasp\_blas\_dbsr\_mvx\_agg ( dBSRmat \* *A*, REAL \* *x*, REAL \* *y* )

Compute  $y := A*x$ , where each small block matrices of *A* is an identity matrix.

## Parameters

<i>A</i>	Pointer to the <a href="#">dBSRmat</a> matrix
<i>x</i>	Pointer to the array <i>x</i>
<i>y</i>	Pointer to the array <i>y</i>

## Author

Xiaozhe Hu

## Date

01/02/2014

## Note

Works for general nb (Xiaozhe)

Definition at line 2641 of file blas\_bsr.c.

10.11.2.8 void fasp\_blas\_dbsr\_rap ( dBSRmat \* *R*, dBSRmat \* *A*, 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 4895 of file blas\_bsr.c.

10.11.2.9 void fasp\_blas\_dbsr\_rap1 ( dBSRmat \* *R*, dBSRmat \* *A*, dBSRmat \* *P*, dBSRmat \* *B* )

dBSRmat sparse matrix multiplication  $B=R*A*P$

## Parameters

<i>R</i>	Pointer to the dBSRmat matrix
<i>A</i>	Pointer to the dBSRmat matrix
<i>P</i>	Pointer to the dBSRmat matrix
<i>B</i>	Pointer to dBSRmat 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 4711 of file blas\_bsr.c.

10.11.2.10 void fasp\_blas\_dbsr\_rap\_agg ( dBSRmat \* *R*, dBSRmat \* *A*, 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 dBSRmat matrix
<i>A</i>	Pointer to the dBSRmat matrix
<i>P</i>	Pointer to the dBSRmat matrix
<i>B</i>	Pointer to dBSRmat matrix equal to $R*A*P$ (output)

## Author

Xiaozhe Hu

## 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 5160 of file blas\_bsr.c.

## 10.12 blas\_csr.c File Reference

BLAS2 operations for `dCSRmat` matrices.

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

### Functions

- `INT fasp_blas_dcsr_add` (`dCSRmat *A`, const `REAL` `alpha`, `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_m xv` (`dCSRmat *A`, `REAL *x`, `REAL *y`)  
*Matrix-vector multiplication  $y = A * x$ .*
- void `fasp_blas_dcsr_mxv_agg` (`dCSRmat *A`, `REAL *x`, `REAL *y`)  
*Matrix-vector multiplication  $y = A * x$ , where the entries of A are all ones.*
- void `fasp_blas_dcsr_aAxy` (const `REAL` `alpha`, `dCSRmat *A`, `REAL *x`, `REAL *y`)  
*Matrix-vector multiplication  $y = \alpha * A * x + y$ .*
- void `fasp_blas_dcsr_aAxy_agg` (const `REAL` `alpha`, `dCSRmat *A`, `REAL *x`, `REAL *y`)  
*Matrix-vector multiplication  $y = \alpha * A * x + y$  (the entries of A are all ones)*
- `REAL fasp_blas_dcsr_vmv` (`dCSRmat *A`, `REAL *x`, `REAL *y`)  
*vector-Matrix-vector multiplication  $\alpha = y' * A * x$*
- void `fasp_blas_dcsr_m xm` (`dCSRmat *A`, `dCSRmat *B`, `dCSRmat *C`)  
*Sparse matrix multiplication  $C = A * B$ .*
- void `fasp_blas_dcsr_rap` (`dCSRmat *R`, `dCSRmat *A`, `dCSRmat *P`, `dCSRmat *RAP`)  
*Triple sparse matrix multiplication  $B = R * A * P$ .*
- void `fasp_blas_dcsr_rap_agg` (`dCSRmat *R`, `dCSRmat *A`, `dCSRmat *P`, `dCSRmat *RAP`)  
*Triple sparse matrix multiplication  $B = R * A * P$ .*
- void `fasp_blas_dcsr_rap_agg1` (`dCSRmat *R`, `dCSRmat *A`, `dCSRmat *P`, `dCSRmat *B`)  
*Triple sparse matrix multiplication  $B = R * A * P$  (nonzero entries of R and P are ones)*
- void `fasp_blas_dcsr_ptap` (`dCSRmat *Pt`, `dCSRmat *A`, `dCSRmat *P`, `dCSRmat *Ac`)  
*Triple sparse matrix multiplication  $B = P' * A * P$ .*
- void `fasp_blas_dcsr_rap4` (`dCSRmat *R`, `dCSRmat *A`, `dCSRmat *P`, `dCSRmat *B`, `INT *icor_ysk`)  
*Triple sparse matrix multiplication  $B = R * A * P$ .*
- void `fasp_blas_dcsr_bandwidth` (`dCSRmat *A`, `INT *bndwith`)  
*Get bandwidth of matrix.*

### 10.12.1 Detailed Description

BLAS2 operations for `dCSRmat` matrices.

**Note**

Sparse functions usually contain three runs. The three runs are all the same but they serve different purposes.

Example: If you do  $c=a+b$ :

- first do a dry run to find the number of non-zeroes in the result 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.

**10.12.2 Function Documentation**

**10.12.2.1** void fasp\_blas\_dcsr\_aApy ( const REAL *alpha*, dCSRmat \* *A*, 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 479 of file blas\_csr.c.

**10.12.2.2** void fasp\_blas\_dcsr\_aApy\_agg ( const REAL *alpha*, dCSRmat \* *A*, REAL \* *x*, REAL \* *y* )

Matrix-vector multiplication  $y = \alpha A x + y$  (the entries of A are all ones)

**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 593 of file blas\_csr.c.

10.12.2.3 void fasp\_blas\_dcsr\_add ( dCSRmat \* *A*, const REAL *alpha*, 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 48 of file blas\_csr.c.

10.12.2.4 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 <i>A</i>
<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 201 of file blas\_csr.c.



#### 10.12.2.5 fasp\_blas\_dcsr\_bandwidth ( dCSRmat \* A, INT \* bndwidth )

Get bandwidth of matrix.

## Parameters

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

## Author

Zheng Li

## Date

03/22/2015

Definition at line 1999 of file blas\_csr.c.

10.12.2.6 void fasp\_blas\_dcsr\_mxm ( [dCSRmat](#) \* *A*, [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

## Note

This fct will be replaced! –Chensong

Definition at line 759 of file blas\_csr.c.

10.12.2.7 void fasp\_blas\_dcsr\_mxv ( [dCSRmat](#) \* *A*, [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 225 of file blas\_csr.c.

10.12.2.8 void fasp\_blas\_dcsr\_mxv\_agg ( dCSRmat \* A, REAL \* x, REAL \* y )

Matrix-vector multiplication  $y = A*x$ , where the entries of A are all ones.

## Parameters

A	Pointer to dCSRmat matrix A
x	Pointer to array x
y	Pointer to array y

## Author

Xiaozhe Hu

## Date

02/22/2011

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

Definition at line 423 of file blas\_csr.c.

10.12.2.9 void fasp\_blas\_dcsr\_ptap ( dCSRmat \* Pt, dCSRmat \* A, dCSRmat \* P, dCSRmat \* Ac )

Triple sparse matrix multiplication  $B=P'*A*P$ .

## Parameters

Pt	Pointer to the restriction matrix
A	Pointer to the fine coefficient matrix
P	Pointer to the prolongation matrix
Ac	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 ltx CSR format. In ltx 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 1596 of file blas\_csr.c.

10.12.2.10 `void fasp_blas_dcsr_rap ( dCSRmat * R, dCSRmat * A, 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 866 of file blas\_csr.c.

10.12.2.11 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 1698 of file blas\_csr.c.

10.12.2.12 void fasp\_blas\_dcsr\_rap\_agg ( [dCSRmat](#) \* *R*, [dCSRmat](#) \* *A*, [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

Xiaozhe Hu

## 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 1148 of file blas\_csr.c.

10.12.2.13 `void fasp_blas_dcsr_rap_agg1 ( dCSRmat * R, dCSRmat * A, dCSRmat * P, dCSRmat * B )`

Triple sparse matrix multiplication  $B=R*A*P$  (nonzero entries of R and P are ones)

## Parameters

<i>R</i>	Pointer to the <a href="#">dCSRmat</a> matrix R
<i>A</i>	Pointer to the <a href="#">dCSRmat</a> matrix A
<i>P</i>	Pointer to the <a href="#">dCSRmat</a> matrix P
<i>B</i>	Pointer to <a href="#">dCSRmat</a> matrix equal to $R*A*P$

## Author

Xiaozhe Hu

## Date

02/21/2011

## Note

Ref. R.E. Bank and C.C. Douglas. SMMP: Sparse Matrix Multiplication Package. Advances in Computational Mathematics, 1 (1993), pp. 127-137.

Definition at line 1413 of file blas\_csr.c.

10.12.2.14 `REAL fasp_blas_dcsr_vmv ( dCSRmat * A, REAL * x, 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 704 of file blas\_csrl.c.

## 10.13 blas\_csrl.c File Reference

BLAS2 operations for [dCSRmat](#) matrices.

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

## Functions

- void [fasp\\_blas\\_dcsrl\\_mnv](#) ([dCSRmat](#) \*A, [REAL](#) \*x, [REAL](#) \*y)  
*Compute  $y = A*x$  for a sparse matrix in CSRL format.*

### 10.13.1 Detailed Description

BLAS2 operations for [dCSRmat](#) matrices.

## Note

For details of CSRL format, refer to "Optimizaing sparse matrix vector product computations using unroll and jam" by John Mellor-Crummey and John Garvin, Tech Report Rice Univ, Aug 2002.

### 10.13.2 Function Documentation

10.13.2.1 void [fasp\\_blas\\_dcsrl\\_mnv](#) ( [dCSRmat](#) \* A, [REAL](#) \* x, [REAL](#) \* y )

Compute  $y = A*x$  for a sparse matrix in CSRL format.

## Parameters

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

$x$	Pointer to REAL array of vector $x$
$y$	Pointer to REAL array of vector $y$

**Date**

2011/01/07

Definition at line 28 of file blas\_csrl.c.

**10.14 blas\_smat.c File Reference**BLAS2 operations for *small* dense matrices.

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

**Functions**

- void [fasp\\_blas\\_smat\\_axm](#) (REAL \*a, const INT n, const REAL alpha)
 

Compute  $\alpha * a$ , store in  $a$ .
- void [fasp\\_blas\\_smat\\_add](#) (REAL \*a, 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](#) (REAL \*a, 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](#) (REAL \*a, 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](#) (REAL \*a, 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](#) (REAL \*a, 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](#) (REAL \*a, REAL \*b, REAL \*c, const INT n)
 

Compute the product of a small full matrix  $a$  and a array  $b$ , stored in  $c$ .
- void [fasp\\_blas\\_smat\\_m ul\\_nc2](#) (REAL \*a, REAL \*b, REAL \*c)
 

Compute the matrix product of two 2\* matrices  $a$  and  $b$ , stored in  $c$ .
- void [fasp\\_blas\\_smat\\_m ul\\_nc3](#) (REAL \*a, REAL \*b, REAL \*c)
 

Compute the matrix product of two 3\*3 matrices  $a$  and  $b$ , stored in  $c$ .
- void [fasp\\_blas\\_smat\\_m ul\\_nc5](#) (REAL \*a, REAL \*b, REAL \*c)
 

Compute the matrix product of two 5\*5 matrices  $a$  and  $b$ , stored in  $c$ .
- void [fasp\\_blas\\_smat\\_m ul\\_nc7](#) (REAL \*a, REAL \*b, REAL \*c)
 

Compute the matrix product of two 7\*7 matrices  $a$  and  $b$ , stored in  $c$ .
- void [fasp\\_blas\\_smat\\_m ul](#) (REAL \*a, 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, REAL \*x, REAL \*y, REAL \*z)
 

$z = a * x + y$
- void [fasp\\_blas\\_array\\_axpyz\\_nc3](#) (const REAL a, REAL \*x, REAL \*y, REAL \*z)
 

$z = a * x + y$



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

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

$z = a*x + y$
- void `fasp_blas_array_axpy_nc2` (const `REAL` a, `REAL` \*x, `REAL` \*y)
 

$y = a*x + y$ , the length of x and y is 2
- void `fasp_blas_array_axpy_nc3` (const `REAL` a, `REAL` \*x, `REAL` \*y)
 

$y = a*x + y$ , the length of x and y is 3
- void `fasp_blas_array_axpy_nc5` (const `REAL` a, `REAL` \*x, `REAL` \*y)
 

$y = a*x + y$ , the length of x and y is 5
- void `fasp_blas_array_axpy_nc7` (const `REAL` a, `REAL` \*x, `REAL` \*y)
 

$y = a*x + y$ , the length of x and y is 7
- void `fasp_blas_smat_ypAx_nc2` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y + Ax$ , where 'A' is a 2\*2 dense matrix.
- void `fasp_blas_smat_ypAx_nc3` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y + Ax$ , where 'A' is a 3\*3 dense matrix.
- void `fasp_blas_smat_ypAx_nc5` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y + Ax$ , where 'A' is a 5\*5 dense matrix.
- void `fasp_blas_smat_ypAx_nc7` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y + Ax$ , where 'A' is a 7\*7 dense matrix.
- void `fasp_blas_smat_ypAx` (`REAL` \*A, `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` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y - Ax$ , where 'A' is a n\*n dense matrix.
- void `fasp_blas_smat_ymAx_nc3` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y - Ax$ , where 'A' is a n\*n dense matrix.
- void `fasp_blas_smat_ymAx_nc5` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y - Ax$ , where 'A' is a n\*n dense matrix.
- void `fasp_blas_smat_ymAx_nc7` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y := y - Ax$ , where 'A' is a 7\*7 dense matrix.
- void `fasp_blas_smat_ymAx` (`REAL` \*A, `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, `REAL` \*A, `REAL` \*x, const `REAL` beta, `REAL` \*y, const `INT` n)
 

Compute  $y := \alpha * A * x + \beta * y$ .
- void `fasp_blas_smat_ymAx_ns2` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y_s := y_s - Ass * x_s$ , where 'A' is a 2\*2 dense matrix, Ass is its saturaton part 1\*1.
- void `fasp_blas_smat_ymAx_ns3` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y_s := y_s - Ass * x_s$ , where 'A' is a 3\*3 dense matrix, Ass is its saturaton part 2\*2.
- void `fasp_blas_smat_ymAx_ns5` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y_s := y_s - Ass * x_s$ , where 'A' is a 5\*5 dense matrix, Ass is its saturaton part 4\*4.
- void `fasp_blas_smat_ymAx_ns7` (`REAL` \*A, `REAL` \*x, `REAL` \*y)
 

Compute  $y_s := y_s - Ass * x_s$ , where 'A' is a 7\*7 dense matrix, Ass is its saturaton part 6\*6.
- void `fasp_blas_smat_ymAx_ns` (`REAL` \*A, `REAL` \*x, `REAL` \*y, const `INT` n)
 

Compute  $y_s := y_s - Ass * x_s$ , where 'A' is a n\*n dense matrix, Ass is its saturaton part (n-1)\*(n-1).

### 10.14.1 Detailed Description

BLAS2 operations for *small* dense matrices.

#### Warning

The routines are designed for full matrices only!

### 10.14.2 Function Documentation

**10.14.2.1** void fasp\_blas\_array\_axpy\_nc2 ( const REAL a, REAL \* x, REAL \* y )

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

#### Parameters

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

#### Author

Xiaozhe Hu

#### Date

18/11/2011

Definition at line 685 of file blas\_smat.c.

**10.14.2.2** void fasp\_blas\_array\_axpy\_nc3 ( const REAL a, REAL \* x, REAL \* y )

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

#### Parameters

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

#### Author

Xiaozhe Hu, Shiquan Zhang

#### Date

05/01/2010

Definition at line 708 of file blas\_smat.c.

**10.14.2.3** void fasp\_blas\_array\_axpy\_nc5 ( const REAL a, REAL \* x, REAL \* y )

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

## Parameters

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

## Author

Xiaozhe Hu, Shiquan Zhang

## Date

05/01/2010

Definition at line 737 of file blas\_smat.c.

10.14.2.4 void fasp\_blas\_array\_axpy\_nc7 ( const REAL *a*, REAL \* *x*, REAL \* *y* )

$y = a * x + y$ , the length of *x* and *y* is 7

## Parameters

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

## Author

Xiaozhe Hu, Shiquan Zhang

## Date

05/01/2010

Definition at line 784 of file blas\_smat.c.

10.14.2.5 void fasp\_blas\_array\_axpyz\_nc2 ( const REAL *a*, REAL \* *x*, 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 500 of file blas\_smat.c.

10.14.2.6 void fasp\_blas\_array\_axpyz\_nc3 ( const REAL a, REAL \* x, REAL \* y, REAL \* z )

$z = a * x + y$

**Parameters**

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

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

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

Definition at line 527 of file blas\_smat.c.

10.14.2.7 void fasp\_blas\_array\_axpyz\_nc5 ( const REAL a, REAL \* x, REAL \* y, REAL \* z )

$z = a * x + y$

**Parameters**

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

**Author**

Xiaozhe Hu, Shiquan Zhang

**Date**

05/01/2010

**Note**

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

Definition at line 560 of file blas\_smat.c.

10.14.2.8 void fasp\_blas\_array\_axpyz\_nc7 ( const REAL a, REAL \* x, 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 611 of file blas\_smat.c.

10.14.2.9 void fasp\_blas\_smat\_aAxpby ( const REAL *alpha*, REAL \* *A*, 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

## Date

2010/10/25

Definition at line 1308 of file blas\_smat.c.

10.14.2.10 void fasp\_blas\_smat\_add ( REAL \* *a*, 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

## Date

05/26/2014

Definition at line 54 of file blas\_smat.c.

10.14.2.11 void fasp\_blas\_smat\_axm ( REAL \* *a*, const INT *n*, const REAL *alpha* )

Compute  $\alpha * a$ , store in *a*.

## 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

## Date

05/26/2014

Definition at line 26 of file blas\_smat.c.

10.14.2.12 void fasp\_blas\_smat\_mul ( REAL \* *a*, 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 448 of file blas\_smat.c.

**10.14.2.13** void fasp\_blas\_smat\_mul\_nc2 ( REAL \* *a*, 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 233 of file blas\_smat.c.

**10.14.2.14** void fasp\_blas\_smat\_mul\_nc3 ( REAL \* *a*, 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 262 of file blas\_smat.c.

**10.14.2.15** void fasp\_blas\_smat\_mul\_nc5 ( REAL \* *a*, 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 299 of file blas\_smat.c.

10.14.2.16 void fasp\_blas\_smat\_mul\_nc7 ( REAL \* *a*, 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 358 of file blas\_smat.c.

10.14.2.17 void fasp\_blas\_smat\_mxv ( REAL \* *a*, 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 183 of file blas\_smat.c.

10.14.2.18 `void fasp_blas_smat_m xv_nc2 ( REAL * a, 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 83 of file blas\_smat.c.

10.14.2.19 void fasp\_blas\_smat\_mnv\_nc3 ( REAL \* *a*, 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 105 of file blas\_smat.c.

10.14.2.20 void fasp\_blas\_smat\_mnv\_nc5 ( REAL \* *a*, 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 128 of file blas\_smat.c.

10.14.2.21 `void fasp_blas_smat_m xv_nc7 ( REAL * a, 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 154 of file blas\_smat.c.

10.14.2.22 void fasp\_blas\_smat\_ymAx ( REAL \* *A*, 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 <i>n</i>
<i>y</i>	Pointer to the REAL array with length <i>n</i>
<i>n</i>	the dimension of the dense matrix

## Author

Zhiyang Zhou, Xiaozhe Hu

## Date

2010/10/25

Definition at line 1207 of file blas\_smat.c.

10.14.2.23 void fasp\_blas\_smat\_ymAx\_nc2 ( REAL \* *A*, REAL \* *x*, REAL \* *y* )

Compute  $y := y - Ax$ , where '*A*' is a  $n \times n$  dense matrix.

## Parameters

<i>A</i>	Pointer to the 2*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 1077 of file blas\_smat.c.

**10.14.2.24** void fasp\_blas\_smat\_yMAx\_nc3 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y := y - Ax$ , where 'A' is a  $n \times n$  dense matrix.

**Parameters**

A	Pointer to the 3*3 dense matrix
x	Pointer to the REAL array with length 3
y	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 1105 of file blas\_smat.c.

**10.14.2.25** void fasp\_blas\_smat\_yMAx\_nc5 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y := y - Ax$ , where 'A' is a  $n \times n$  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**

Xiaozhe Hu, Zhiyang Zhou

**Date**

01/06/2011

**Note**

Works for 5-component

Definition at line 1135 of file blas\_smat.c.

**10.14.2.26** void fasp\_blas\_smat\_yMAx\_nc7 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y := y - Ax$ , where 'A' is a  $7 \times 7$  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

Xiaozhe Hu, Zhiyang Zhou

## Date

01/06/2011

## Note

Works for 7-component

Definition at line 1169 of file blas\_smat.c.

10.14.2.27 void fasp\_blas\_smat\_yMAx\_ns ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$ , const INT  $n$  )

Compute  $y_s := y_s - A s s * x_s$ , where ' $A$ ' is a  $n*n$  dense matrix,  $A s s$  is its saturaton part  $(n-1)*(n-1)$ .

## Parameters

$A$	Pointer to the $n*n$ dense matrix
$x$	Pointer to the REAL array with length $n-1$
$y$	Pointer to the REAL array with length $n-1$
$n$	the dimension of the dense matrix

## Author

Xiaozhe Hu

## Date

2010/10/25

## Note

Only for block smoother for saturation block without explicitly use saturation block!!

Definition at line 1482 of file blas\_smat.c.

10.14.2.28 void fasp\_blas\_smat\_yMAx\_ns2 ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$  )

Compute  $y_s := y_s - A s s * x_s$ , where ' $A$ ' is a  $2*2$  dense matrix,  $A s s$  is its saturaton part  $1*1$ .

## Parameters

$A$	Pointer to the 2*2 dense matrix
$x$	Pointer to the REAL array with length 1
$y$	Pointer to the REAL array with length 1

## Author

Xiaozhe Hu

## Date

2011/11/18

## Note

Works for 2-component (Xiaozhe) Only for block smoother for saturation block without explicitly use saturation block!!

Definition at line 1358 of file blas\_smat.c.

10.14.2.29 void fasp\_blas\_smat\_ymAx\_ns3 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y_s := y_s - A_{ss}x_s$ , where 'A' is a 3\*3 dense matrix,  $A_{ss}$  is its saturation part 2\*2.

## Parameters

$A$	Pointer to the 3*3 dense matrix
$x$	Pointer to the REAL array with length 2
$y$	Pointer to the REAL array with length 2

## Author

Xiaozhe Hu

## Date

2010/10/25

## Note

Works for 3-component (Xiaozhe) Only for block smoother for saturation block without explicitly use saturation block!!

Definition at line 1382 of file blas\_smat.c.

10.14.2.30 void fasp\_blas\_smat\_ymAx\_ns5 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y_s := y_s - A_{ss}x_s$ , where 'A' is a 5\*5 dense matrix,  $A_{ss}$  is its saturation part 4\*4.



## Parameters

$A$	Pointer to the 5*5 dense matrix
$x$	Pointer to the REAL array with length 4
$y$	Pointer to the REAL array with length 4

## Author

Xiaozhe Hu

## Date

2010/10/25

## Note

Works for 5-component (Xiaozhe) Only for block smoother for saturation block without explicitly use saturation block!!

Definition at line 1410 of file blas\_smat.c.

10.14.2.31 void fasp\_blas\_smat\_yMAx\_ns7 ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$  )

Compute  $y := y - A s s * x s$ , where ' $A$ ' is a 7\*7 dense matrix,  $A s s$  is its saturation part 6\*6.

## Parameters

$A$	Pointer to the 7*7 dense matrix
$x$	Pointer to the REAL array with length 6
$y$	Pointer to the REAL array with length 6

## Author

Xiaozhe Hu

## Date

2010/10/25

## Note

Works for 7-component (Xiaozhe) Only for block smoother for saturation block without explicitly use saturation block!!

Definition at line 1444 of file blas\_smat.c.

10.14.2.32 void fasp\_blas\_smat\_yPAx ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$ , const INT  $n$  )

Compute  $y := y + A x$ , where ' $A$ ' is a  $n * n$  dense matrix.

## Parameters

$A$	Pointer to the $n \times n$ dense matrix
$x$	Pointer to the REAL array with length $n$
$y$	Pointer to the REAL array with length $n$
$n$	Dimension of the dense matrix

## Author

Zhiyang Zhou

## Date

2010/10/25

Definition at line 976 of file blas\_smat.c.

10.14.2.33 void fasp\_blas\_smat\_ypAx\_nc2 ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$  )

Compute  $y := y + Ax$ , where ' $A$ ' is a  $2 \times 2$  dense matrix.

## Parameters

$A$	Pointer to the $3 \times 3$ dense matrix
$x$	Pointer to the REAL array with length 3
$y$	Pointer to the REAL array with length 3

## Author

Xiaozhe Hu

## Date

2011/11/18

Definition at line 857 of file blas\_smat.c.

10.14.2.34 void fasp\_blas\_smat\_ypAx\_nc3 ( REAL \*  $A$ , REAL \*  $x$ , REAL \*  $y$  )

Compute  $y := y + Ax$ , where ' $A$ ' is a  $3 \times 3$  dense matrix.

## Parameters

$A$	Pointer to the $3 \times 3$ dense matrix
$x$	Pointer to the REAL array with length 3
$y$	Pointer to the REAL array with length 3

## Author

Zhiyang Zhou, Xiaozhe Hu

## Date

2010/10/25

Definition at line 883 of file blas\_smat.c.

10.14.2.35 void fasp\_blas\_smat\_ypAx\_nc5 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y := y + Ax$ , where 'A' is a 5\*5 dense matrix.

## Parameters

<i>A</i>	Pointer to the 5*5 dense matrix
<i>x</i>	Pointer to the REAL array with length 5
<i>y</i>	Pointer to the REAL array with length 5

## Author

Zhiyang Zhou, Xiaozhe Hu

## Date

2010/10/25

Definition at line 910 of file blas\_smat.c.

10.14.2.36 void fasp\_blas\_smat\_ypAx\_nc7 ( REAL \* A, REAL \* x, REAL \* y )

Compute  $y := y + Ax$ , where 'A' is a 7\*7 dense matrix.

## Parameters

<i>A</i>	Pointer to the 7*7 dense matrix
<i>x</i>	Pointer to the REAL array with length 7
<i>y</i>	Pointer to the REAL array with length 7

## Author

Zhiyang Zhou, Xiaozhe Hu

## Date

2010/10/25

Definition at line 941 of file blas\_smat.c.

## 10.15 blas\_str.c File Reference

BLAS2 operations for [dSTRmat](#) matrices.

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

## Functions

- void [fasp\\_blas\\_dstr\\_aAxy](#) (const [REAL](#) alpha, [dSTRmat](#) \*A, [REAL](#) \*x, [REAL](#) \*y)  
*Matrix-vector multiplication  $y = \alpha * A * x + y$ .*
- void [fasp\\_blas\\_dstr\\_mxv](#) ([dSTRmat](#) \*A, [REAL](#) \*x, [REAL](#) \*y)  
*Matrix-vector multiplication  $y = A * x$ .*
- [INT](#) [fasp\\_dstr\\_diagscale](#) ([dSTRmat](#) \*A, [dSTRmat](#) \*B)  
 *$B = D^{\{-1\}} A$ .*

### 10.15.1 Detailed Description

BLAS2 operations for [dSTRmat](#) matrices.

### 10.15.2 Function Documentation

10.15.2.1 void fasp\_blas\_dstr\_aApy ( const REAL *alpha*, dSTRmat \* *A*, REAL \* *x*, REAL \* *y* )

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

Parameters

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

Author

Zhiyang Zhou, Xiaozhe Hu, Shiquan Zhang

Date

2010/10/15

Definition at line 47 of file blas\_str.c.

10.15.2.2 void fasp\_blas\_dstr\_mxv ( dSTRmat \* *A*, 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 117 of file blas\_str.c.

10.15.2.3 INT fasp\_dstr\_diagscale ( dSTRmat \* *A*, dSTRmat \* *B* )

$B = D^{-1}A$ .

## Parameters

<i>A</i>	Pointer to a 'dSTRmat' type matrix A
<i>B</i>	Pointer to a 'dSTRmat' type matrix B

## Author

Shiquan Zhang

## Date

2010/10/15

Modified by Chunsheng Feng, Zheng Li

## Date

08/30/2012

Definition at line 142 of file blas\_str.c.

## 10.16 blas\_vec.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, [dvector](#) \*x, [dvector](#) \*y)  
 $y = a*x + y$
- void [fasp\\_blas\\_dvec\\_axpyz](#) (const [REAL](#) a, [dvector](#) \*x, [dvector](#) \*y, [dvector](#) \*z)  
 $z = a*x + y$ , *z is a third vector (z is cleared)*
- [REAL fasp\\_blas\\_dvec\\_dotprod](#) ([dvector](#) \*x, [dvector](#) \*y)  
*Inner product of two vectors (x,y)*
- [REAL fasp\\_blas\\_dvec\\_relerr](#) ([dvector](#) \*x, [dvector](#) \*y)  
*Relative error of two dvector x and y.*
- [REAL fasp\\_blas\\_dvec\\_norm1](#) ([dvector](#) \*x)  
*L1 norm of dvector x.*
- [REAL fasp\\_blas\\_dvec\\_norm2](#) ([dvector](#) \*x)  
*L2 norm of dvector x.*
- [REAL fasp\\_blas\\_dvec\\_norminf](#) ([dvector](#) \*x)  
*Linf norm of dvector x.*

#### 10.16.1 Detailed Description

BLAS1 operations for vectors.

## 10.16.2 Function Documentation

10.16.2.1 void fasp\_blas\_dvec\_axpy ( const REAL a, 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

07/01/209

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

Definition at line 33 of file blas\_vec.c.

10.16.2.2 void fasp\_blas\_dvec\_axpyz ( const REAL a, dvector \* x, 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

Date

05/23/2012

Definition at line 85 of file blas\_vec.c.

10.16.2.3 REAL fasp\_blas\_dvec\_dotprod ( dvector \* x, dvector \* y )

Inner product of two vectors (x,y)

**Parameters**

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

**Returns**

Inner product

**Author**

Chensong Zhang

**Date**

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue

**Date**

05/23/2012

Definition at line 121 of file blas\_vec.c.

**10.16.2.4 REAL fasp\_blas\_dvec\_norm1 ( 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 222 of file blas\_vec.c.

**10.16.2.5 REAL fasp\_blas\_dvec\_norm2 ( 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

## Date

05/23/2012

Definition at line 265 of file blas\_vec.c.

#### 10.16.2.6 **REAL** fasp\_blas\_dvec\_norminf ( dvector \* $x$ )

Linf norm of dvector  $x$ .

## Parameters

$x$	Pointer to dvector $x$
-----	------------------------

## Returns

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

## Author

Chensong Zhang

## Date

07/01/2009

Definition at line 305 of file blas\_vec.c.

#### 10.16.2.7 **REAL** fasp\_blas\_dvec\_relerr ( dvector \* $x$ , 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

**Date**

05/23/2012

Definition at line 167 of file blas\_vec.c.

## 10.17 checkmat.c File Reference

Check matrix properties.

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

**Functions**

- [INT fasp\\_check\\_diagpos \(dCSRmat \\*A\)](#)  
*Check positivity of diagonal entries of a CSR sparse matrix.*
- [SHORT fasp\\_check\\_diagzero \(dCSRmat \\*A\)](#)  
*Check wether a CSR sparse matrix has diagonal entries that are very close to zero.*
- [INT fasp\\_check\\_diagdom \(dCSRmat \\*A\)](#)  
*Check whether a matrix is diagonal dominant.*
- [INT fasp\\_check\\_symm \(dCSRmat \\*A\)](#)  
*Check symmetry of a sparse matrix of CSR format.*
- [SHORT fasp\\_check\\_dCSRmat \(dCSRmat \\*A\)](#)  
*Check whether an [dCSRmat](#) matrix is valid or not.*
- [SHORT fasp\\_check\\_iCSRmat \(iCSRmat \\*A\)](#)  
*Check whether an [iCSRmat](#) matrix is valid or not.*

### 10.17.1 Detailed Description

Check matrix properties.

### 10.17.2 Function Documentation

#### 10.17.2.1 **SHORT** fasp\_check\_dCSRmat ( dCSRmat \* A )

Check whether an [dCSRmat](#) matrix is valid or not.

Parameters

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

Author

Shuo Zhang

Date

03/29/2009

Definition at line 275 of file checkmat.c.

#### 10.17.2.2 **INT** fasp\_check\_diagdom ( dCSRmat \* A )

Check whether a matrix is diagonal dominant.

**INT** fasp\_check\_diagdom ([dCSRmat](#) \*A)

Parameters

<b>A</b>	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 108 of file checkmat.c.

### 10.17.2.3 INT fasp\_check\_diagpos ( 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 27 of file checkmat.c.

**10.17.2.4 SHORT fasp\_check\_diagzero ( 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 64 of file checkmat.c.

**10.17.2.5 SHORT fasp\_check\_iCSRmat ( iCSRmat \* *A* )**

Check whether an [iCSRmat](#) matrix is valid or not.

## Parameters

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

## Author

Shuo Zhang

## Date

03/29/2009

Definition at line 309 of file checkmat.c.

#### 10.17.2.6 INT fasp\_check\_symm ( dCSRmat \* A )

Check symmetry of a sparse matrix of CSR format.

## Parameters

<i>A</i>	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 153 of file checkmat.c.

## 10.18 coarsening\_cr.c File Reference

Coarsening with Brannick-Falgout strategy.

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

### Functions

- [INT fasp\\_amg\\_coarsening\\_cr](#) (const [INT](#) *i\_0*, const [INT](#) *i\_n*, [dCSRmat](#) \**A*, [ivector](#) \**vertices*, [AMG\\_param](#) \**param*)  
*CR coarsening.*

#### 10.18.1 Detailed Description

Coarsening with Brannick-Falgout strategy.

#### 10.18.2 Function Documentation

**10.18.2.1** [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

Modified by Chunsheng Feng, Zheng Li on 10/14/2012 CR STAGES

Definition at line 42 of file coarsening\_cr.c.

**10.19 coarsening\_rs.c File Reference**

Coarsening with a modified Ruge-Stuben strategy.

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

**Functions**

- [SHORT fasp\\_amg\\_coarsening\\_rs](#) ([dCSRmat](#) \*A, [ivector](#) \*vertices, [dCSRmat](#) \*P, [iCSRmat](#) \*S, [AMG\\_param](#) \*param)

*Standard and aggressive coarsening schemes.*

**10.19.1 Detailed Description**

Coarsening with a modified Ruge-Stuben strategy.

**Note**

Ref 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.

ATTENTION: Do NOT use auto-indentation in this file!!!



## 10.19.2 Function Documentation

10.19.2.1 **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 61 of file coarsening\_rs.c.

## 10.20 `convert.c` File Reference

Some utilities for format conversion.

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

## Functions

- unsigned long [fasp\\_aux\\_change\\_endian4](#) (unsigned long x)  
*Swap order for different endian systems.*
- double [fasp\\_aux\\_change\\_endian8](#) (double x)  
*Swap order for different endian systems.*
- double [fasp\\_aux\\_bbyteToldouble](#) (unsigned char bytes[])  
*Swap order of double-precision float for different endian systems.*
- [INT endian\\_convert\\_int](#) (const [INT](#) inum, const [INT](#) ilength, const [INT](#) endianflag)  
*Swap order of an INT number.*
- [REAL endian\\_convert\\_real](#) (const [REAL](#) rnum, const [INT](#) vlength, const [INT](#) endianflag)  
*Swap order of a REAL number.*

### 10.20.1 Detailed Description

Some utilities for format conversion.

### 10.20.2 Function Documentation

#### 10.20.2.1 INT endian\_convert\_int ( const INT *inum*, const INT *ilength*, const INT *endianflag* )

Swap order of an INT number.

##### Parameters

<i>inum</i>	An INT value
<i>ilength</i>	Length of INT: 2 for short, 4 for int, 8 for long
<i>endianflag</i>	If <i>endianflag</i> = 1, it returns <i>inum</i> itself If <i>endianflag</i> = 2, it returns the swapped <i>inum</i>

##### Returns

Value of *inum* or swapped *inum*

##### Author

Ziteng Wang

##### Date

2012-12-24

Definition at line 105 of file convert.c.

#### 10.20.2.2 REAL endian\_convert\_real ( const REAL *rnum*, const INT *ilength*, const INT *endianflag* )

Swap order of a REAL number.

##### Parameters

<i>rnum</i>	An REAL value
<i>ilength</i>	Length of INT: 2 for short, 4 for int, 8 for long
<i>endianflag</i>	If <i>endianflag</i> = 1, it returns <i>rnum</i> itself If <i>endianflag</i> = 2, it returns the swapped <i>rnum</i>

##### Returns

Value of *rnum* or swapped *rnum*

##### Author

Ziteng Wang

##### Date

2012-12-24

Definition at line 137 of file convert.c.

10.20.2.3    `double fasp_aux_bbyteToldouble ( 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 74 of file convert.c.

**10.20.2.4 unsigned long fasp\_aux\_change\_endian4 ( unsigned long x )**

Swap order for different endian systems.

**Parameters**

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

**Returns**

Unsigend long ineger after swapping

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 25 of file convert.c.

**10.20.2.5 double fasp\_aux\_change\_endian8 ( double x )**

Swap order for different endian systems.

**Parameters**

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

**Returns**

Unsigend long ineger after swapping

**Author**

Chensong Zhang

**Date**

11/16/2009

Definition at line 43 of file convert.c.

## 10.21 doxygen.h File Reference

Main page for Doygen documentation.

### 10.21.1 Detailed Description

Main page for Doygen documentation.

## 10.22 eigen.c File Reference

Subroutines for computing the extreme eigenvalues.

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

### Functions

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

### 10.22.1 Detailed Description

Subroutines for computing the extreme eigenvalues.

### 10.22.2 Function Documentation

#### 10.22.2.1 [REAL fasp\\_dcsr\\_eig](#) ( 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 29 of file eigen.c.

## 10.23 famg.c File Reference

Full AMG method as an iterative solver (main file)

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

**Functions**

- void [fasp\\_solver\\_famg](#) ([dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [AMG\\_param](#) \*param)  
Solve  $Ax=b$  by full AMG.

### 10.23.1 Detailed Description

Full AMG method as an iterative solver (main file)

### 10.23.2 Function Documentation

10.23.2.1 void [fasp\\_solver\\_famg](#) ( [dCSRmat](#) \* A, [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 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**

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 31 of file famg.c.

## 10.24 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 [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.*
- struct [grid2d](#)  
*Two dimensional grid data structure.*
- struct [Link](#)  
*Struct for Links.*
- struct [linked\\_list](#)  
*A linked list node.*

## Macros

- #define [\\_\\_FASP\\_HEADER\\_\\_](#)
- #define [FASP\\_VERSION](#) 1.8  
*For external software package support.*
- #define [FASP\\_USE\\_ILU](#) ON
- #define [DLMALLOC](#) OFF
- #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`
- typedef struct `grid2d` `grid2d`
- typedef `grid2d` \* `pgrid2d`
- typedef const `grid2d` \* `pcgrid2d`
- typedef struct `linked_list` `ListElement`
- typedef `ListElement` \* `LinkList`

## 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`

### 10.24.1 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.

### 10.24.2 Macro Definition Documentation

#### 10.24.2.1 `#define __FASP_HEADER__`

indicate [fasp.h](#) has been included before

Definition at line 29 of file fasp.h.

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

absolute value of a

Definition at line 67 of file fasp.h.

#### 10.24.2.3 `#define DIAGONAL_PREF OFF`

order each row such that diagonal appears first

Definition at line 51 of file fasp.h.

#### 10.24.2.4 `#define DLMALLOC OFF`

use dlmalloc instead of standard malloc

Definition at line 40 of file fasp.h.

#### 10.24.2.5 `#define FASP_GSRB 1`

MG level 0 use RedBlack Gauss Seidel Smoothing

Definition at line 1161 of file fasp.h.

#### 10.24.2.6 `#define FASP_USE_ILU ON`

enable ILU or not

Definition at line 39 of file fasp.h.

#### 10.24.2.7 `#define FASP_VERSION 1.8`

For external software package support.

faspsolver version

Definition at line 38 of file fasp.h.

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

is  $a \geq b$ ?

Definition at line 73 of file fasp.h.

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

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

is  $a > b$ ?

Definition at line 72 of file fasp.h.

#### 10.24.2.10 `#define INT int`

regular integer type: int or long

Definition at line 57 of file fasp.h.

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

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

Definition at line 76 of file fasp.h.

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

is  $a \leq b$ ?

Definition at line 75 of file fasp.h.

#### 10.24.2.13 `#define LONG long`

long integer type

Definition at line 58 of file fasp.h.

#### 10.24.2.14 `#define LONGLONG long long`

long integer type

Definition at line 59 of file fasp.h.

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

is  $a < b$ ?

Definition at line 74 of file fasp.h.

10.24.2.16 **#define MAX( a, b ) (((a)>(b))?(a):(b))**

Definition of max, min, abs.

bigger one in a and b

Definition at line 65 of file fasp.h.

10.24.2.17 **#define MIN( a, b ) (((a)<(b))?(a):(b))**

smaller one in a and b

Definition at line 66 of file fasp.h.

10.24.2.18 **#define NEDMALLOC OFF**

use nedmalloc instead of standard malloc

Definition at line 41 of file fasp.h.

10.24.2.19 **#define PUT\_INT( A ) printf("### DEBUG: %s = %d\n", #A, (A))**

Definition of print command in DEBUG mode.

print an integer

Definition at line 81 of file fasp.h.

10.24.2.20 **#define PUT\_REAL( A ) printf("### DEBUG: %s = %e\n", #A, (A))**

print a real num

Definition at line 82 of file fasp.h.

10.24.2.21 **#define REAL double**

float type

Definition at line 60 of file fasp.h.

10.24.2.22 **#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 49 of file fasp.h.

#### 10.24.2.23 `#define SHORT short`

FASP integer and floating point numbers.

short integer type

Definition at line 56 of file fasp.h.

### 10.24.3 Typedef Documentation

#### 10.24.3.1 `typedef struct dCOOmat dCOOmat`

Sparse matrix of REAL type in COO format

#### 10.24.3.2 `typedef struct dCSRLmat dCSRLmat`

Sparse matrix of REAL type in CSRL format

#### 10.24.3.3 `typedef struct dCSRmat dCSRmat`

Sparse matrix of REAL type in CSR format

#### 10.24.3.4 `typedef struct ddenmat ddenmat`

Dense matrix of REAL type

#### 10.24.3.5 `typedef struct dSTRmat dSTRmat`

Structured matrix of REAL type

#### 10.24.3.6 `typedef struct dvector dvector`

Vector of REAL type

#### 10.24.3.7 `typedef struct grid2d grid2d`

2D grid type for plotting

#### 10.24.3.8 `typedef struct iCOOmat iCOOmat`

Sparse matrix of INT type in COO format

**10.24.3.9 typedef struct iCSRmat iCSRmat**

Sparse matrix of INT type in CSR format

**10.24.3.10 typedef struct idenmat idenmat**

Dense matrix of INT type

**10.24.3.11 typedef struct ivector ivector**

Vector of INT type

**10.24.3.12 typedef ListElement\* LinkList**

List of linkslinked list

Definition at line 1156 of file fasp.h.

**10.24.3.13 typedef struct linked\_list ListElement**

Linked element in list

**10.24.3.14 typedef const grid2d\* pcgrid2d**

Grid in 2d

Definition at line 1110 of file fasp.h.

**10.24.3.15 typedef grid2d\* pgrid2d**

Grid in 2d

Definition at line 1108 of file fasp.h.

**10.24.4 Variable Documentation****10.24.4.1 INT count**

Counter for multiple calls

**10.24.4.2 INT\* IMAP**

Red Black Gs Smoother imap

**10.24.4.3 INT MAXIMAP**

Red Black Gs Smoother max DOFs of reservoir

#### 10.24.4.4 INT nx\_rb

Red Black Gs Smoother Nx

#### 10.24.4.5 INT ny\_rb

Red Black Gs Smoother Ny

#### 10.24.4.6 INT nz\_rb

Red Black Gs Smoother Nz

#### 10.24.4.7 unsigned INT total\_alloc\_count

Total allocated memory amount.

total allocation times

Definition at line 35 of file memory.c.

#### 10.24.4.8 unsigned INT total\_alloc\_mem

total allocated memory

Definition at line 34 of file memory.c.

## 10.25 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 [block\\_dCSRmat](#)  
*Block REAL CSR matrix format.*
- struct [block\\_iCSRmat](#)  
*Block INT CSR matrix format.*
- struct [block\\_dvector](#)  
*Block REAL vector structure.*
- struct [block\\_ivector](#)  
*Block INT vector structure.*
- struct [block\\_Reservoir](#)  
*Block REAL matrix format for reservoir simulation.*
- struct [block\\_BSR](#)



*Block REAL matrix format for reservoir simulation.*

- 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\\_reservoir\\_data](#)

*Data passed to the preconditioner for reservoir simulation problems.*

- struct [precond\\_block\\_data](#)

*Data passed to the preconditioner for block preconditioning for [block\\_dCSRmat](#) format.*

- struct [precond\\_FASP\\_blkoi\\_data](#)

*Data passed to the preconditioner for preconditioning reservoir simulation problems.*

- struct [precond\\_sweeping\\_data](#)

*Data passed to the preconditioner for sweeping preconditioning.*

## Macros

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

- #define [SMOOTHER\\_BLKOil](#) 11

*Definition of specialized smoother types.*

- #define [SMOOTHER\\_SPETEN](#) 19

## Typedefs

- typedef struct [dBSRmat](#) [dBSRmat](#)
- typedef struct [block\\_dCSRmat](#) [block\\_dCSRmat](#)
- typedef struct [block\\_iCSRmat](#) [block\\_iCSRmat](#)
- typedef struct [block\\_dvector](#) [block\\_dvector](#)
- typedef struct [block\\_ivecator](#) [block\\_ivecator](#)
- typedef struct [block\\_Reservoir](#) [block\\_Reservoir](#)
- typedef struct [block\\_BSR](#) [block\\_BSR](#)
- typedef struct [precond\\_block\\_reservoir\\_data](#) [precond\\_block\\_reservoir\\_data](#)

### 10.25.1 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 [precond\\_block\\_↵\\_reservoir\\_data](#). Modified by Xiaozhe Hu on 06/15/2010: modify [precond\\_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 20151011: cosmetics.

## 10.25.2 Macro Definition Documentation

### 10.25.2.1 `#define __FASPBLOCK_HEADER__`

indicate [fasp\\_block.h](#) has been included before  
Definition at line 22 of file `fasp_block.h`.

### 10.25.2.2 `#define SMOOTHER_BLKOil 11`

Definition of specialized smoother types.  
Used in monolithic AMG for black-oil  
Definition at line 27 of file `fasp_block.h`.

### 10.25.2.3 `#define SMOOTHER_SPETEN 19`

Used in monolithic AMG for black-oil  
Definition at line 28 of file `fasp_block.h`.

## 10.25.3 Typedef Documentation

### 10.25.3.1 `typedef struct block_BSR block_BSR`

Block of BSR matrices of REAL type

### 10.25.3.2 `typedef struct block_dCSRmat block_dCSRmat`

Matrix of REAL type in Block CSR format

### 10.25.3.3 `typedef struct block_dvector block_dvector`

Vector of REAL type in Block format

### 10.25.3.4 `typedef struct block_iCSRmat block_iCSRmat`

Matrix of INT type in Block CSR format

### 10.25.3.5 `typedef struct block_ivecator block_ivecator`

Vector of INT type in Block format

### 10.25.3.6 `typedef struct block_Reservoir block_Reservoir`

Special block matrix for Reservoir Simulation

## 10.25.3.7 typedef struct dBSRmat dBSRmat

Matrix of REAL type in BSR format

## 10.25.3.8 typedef struct precondition\_block\_reservoir\_data precondition\_block\_reservoir\_data

Precond data for Reservoir Simulation

## 10.26 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\_bCSR 4
- #define MAT\_bBSR 5
- #define MAT\_CSRL 6
- #define MAT\_SymCSR 7
- #define SOLVER\_DEFAULT 0

*Definition of solver types for iterative methods.*

- #define SOLVER\_CG 1
- #define SOLVER\_BiCGstab 2
- #define SOLVER\_MinRes 3
- #define SOLVER\_GMRES 4
- #define SOLVER\_VGMRES 5
- #define SOLVER\_VFGMRES 6
- #define SOLVER\_GCG 7
- #define SOLVER\_GCR 8
- #define SOLVER\_SCG 11
- #define SOLVER\_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 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 [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 [GOPT](#) -5
  - Type of vertices (DOFs) for coarsening.*
- #define [UNPT](#) -1
- #define [FGPT](#) 0
- #define [CGPT](#) 1
- #define [ISPT](#) 2
- #define [NO\\_ORDER](#) 0

*Definition of smoothing order.*

- #define `CF_ORDER` 1
- #define `USERDEFINED` 0

*Type of ordering for smoothers.*

- #define `CPFIRST` 1
- #define `FPFIRST` -1
- #define `ASCEND` 12
- #define `DESCEND` 21
- #define `BIGREAL` 1e+20

*Some global constants.*

- #define `SMALLREAL` 1e-20
- #define `SMALLREAL2` 1e-40
- #define `MAX_REFINE_LVL` 20
- #define `MAX_AMG_LVL` 20
- #define `MIN_CDOF` 20
- #define `MIN_CRATE` 0.9
- #define `MAX_CRATE` 20.0
- #define `MAX_RESTART` 20
- #define `MAX_STAG` 20
- #define `STAG_RATIO` 1e-4
- #define `OPENMP_HOLDS` 2000

### 10.26.1 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.

### 10.26.2 Macro Definition Documentation

#### 10.26.2.1 #define `AMLI_CYCLE` 3

AMLI-cycle

Definition at line 176 of file `fasp_const.h`.

#### 10.26.2.2 #define `ASCEND` 12

Ascending order

Definition at line 230 of file `fasp_const.h`.

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

Some global constants.

A large real number

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

### 10.26.2.4 `#define CF_ORDER 1`

C/F order smoothing

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

### 10.26.2.5 `#define CGPT 1`

Coarse grid points

Definition at line 215 of file fasp\_const.h.

### 10.26.2.6 `#define CLASSIC_AMG 1`

Definition of AMG types.

classic AMG

Definition at line 161 of file fasp\_const.h.

### 10.26.2.7 `#define COARSE_AC 4`

Aggressive coarsening

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

### 10.26.2.8 `#define COARSE_CR 3`

Compatible relaxation

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

### 10.26.2.9 `#define COARSE_MIS 5`

Aggressive coarsening based on MIS

Definition at line 200 of file fasp\_const.h.

### 10.26.2.10 `#define COARSE_RS 1`

Definition of coarsening types.

Classical

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

**10.26.2.11 #define COARSE\_RSP 2**

Classical, with positive offdiags

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

**10.26.2.12 #define CPFIRST 1**

C-points first order

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

**10.26.2.13 #define DESCEND 21**

Descending order

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

**10.26.2.14 #define ERROR\_ALLOC\_MEM -20**

fail to allocate memory

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

**10.26.2.15 #define ERROR\_AMG\_COARSE\_TYPE -32**

unknown coarsening type

Definition at line 44 of file fasp\_const.h.

**10.26.2.16 #define ERROR\_AMG\_COARSEING -33**

coarsening step failed to complete

Definition at line 45 of file fasp\_const.h.

**10.26.2.17 #define ERROR\_AMG\_INTERP\_TYPE -30**

unknown interpolation type

Definition at line 42 of file fasp\_const.h.

**10.26.2.18 #define ERROR\_AMG\_SMOOTH\_TYPE -31**

unknown smoother type

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

**10.26.2.19 #define ERROR\_DATA\_STRUCTURE -21**

problem with data structures

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



**10.26.2.20 #define ERROR\_DATA\_ZERODIAG -22**

matrix has zero diagonal entries

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

**10.26.2.21 #define ERROR\_DUMMY\_VAR -23**

unexpected input data

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

**10.26.2.22 #define ERROR\_INPUT\_PAR -13**

wrong input argument

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

**10.26.2.23 #define ERROR\_LIC\_TYPE -80**

wrong license type

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

**10.26.2.24 #define ERROR\_MAT\_SIZE -15**

wrong problem size

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

**10.26.2.25 #define ERROR\_MISC -19**

other error

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

**10.26.2.26 #define ERROR\_NUM\_BLOCKS -18**

wrong number of blocks

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

**10.26.2.27 #define ERROR\_OPEN\_FILE -10**

fail to open a file

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

**10.26.2.28 #define ERROR\_QUAD\_DIM -61**

unsupported quadrature dim

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

**10.26.2.29 #define ERROR\_QUAD\_TYPE -60**

unknown quadrature type

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

**10.26.2.30 #define ERROR\_REGRESS -14**

regression test fail

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

**10.26.2.31 #define ERROR\_SOLVER\_EXIT -49**

solver does not quit successfully

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

**10.26.2.32 #define ERROR\_SOLVER\_ILUSETUP -45**

ILU setup error

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

**10.26.2.33 #define ERROR\_SOLVER\_MAXIT -48**

maximal iteration number exceeded

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

**10.26.2.34 #define ERROR\_SOLVER\_MISC -46**

misc solver error during run time

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

**10.26.2.35 #define ERROR\_SOLVER\_PRECTYPE -41**

unknown precond type

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

**10.26.2.36 #define ERROR\_SOLVER\_SOLSTAG -43**

solver's solution is too small

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

**10.26.2.37 #define ERROR\_SOLVER\_STAG -42**

solver stagnates

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

**10.26.2.38 #define ERROR\_SOLVER\_TOLSMALL -44**

solver's tolerance is too small

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

**10.26.2.39 #define ERROR\_SOLVER\_TYPE -40**

unknown solver type

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

**10.26.2.40 #define ERROR\_UNKNOWN -99**

an unknown error type

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

**10.26.2.41 #define ERROR\_WRONG\_FILE -11**

input contains wrong format

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

**10.26.2.42 #define FALSE 0**

logic FALSE

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

**10.26.2.43 #define FASP\_SUCCESS 0**

Definition of return status and error messages.

return from function successfully

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

**10.26.2.44 #define FGPT 0**

Fine grid points

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

**10.26.2.45 #define FPFIRST -1**

F-points first order

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

**10.26.2.46 #define GOPT -5**

Type of vertices (DOFs) for coarsening.

Cannot fit in aggregates

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

**10.26.2.47 #define ILUk 1**

Type of ILU methods.

ILUk

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

**10.26.2.48 #define ILUt 2**

ILUt

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

**10.26.2.49 #define ILUtp 3**

ILUtp

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

**10.26.2.50 #define INTERP\_DIR 1**

Definition of interpolation types.

Direct interpolation

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

**10.26.2.51 #define INTERP\_ENG 3**

energy minimization interpolation

Definition at line 207 of file fasp\_const.h.

**10.26.2.52 #define INTERP\_STD 2**

Standard interpolation

Definition at line 206 of file fasp\_const.h.

**10.26.2.53 #define ISPT 2**

Isolated points

Definition at line 216 of file fasp\_const.h.

**10.26.2.54 #define MAT\_bBSR 5**

block matrix of BSR for bordered systems

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

**10.26.2.55 #define MAT\_bCSR 4**

block matrix of CSR

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

**10.26.2.56 #define MAT\_BSR 2**

block-wise compressed sparse row

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

**10.26.2.57 #define MAT\_CSR 1**

compressed sparse row

Definition at line 90 of file fasp\_const.h.

**10.26.2.58 #define MAT\_CSRL 6**

modified CSR to reduce cache missing

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

**10.26.2.59 #define MAT\_FREE 0**

Definition of matrix format.

matrix-free format: only mxv action

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

**10.26.2.60 #define MAT\_STR 3**

structured sparse matrix

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

**10.26.2.61 #define MAT\_SymCSR 7**

symmetric CSR format

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

10.26.2.62 `#define MAX_AMG_LVL 20`

Maximal AMG coarsening level

Definition at line 240 of file fasp\_const.h.

10.26.2.63 `#define MAX_CRATE 20.0`

Maximal coarsening ratio

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

10.26.2.64 `#define MAX_REFINE_LVL 20`

Maximal refinement level

Definition at line 239 of file fasp\_const.h.

10.26.2.65 `#define MAX_RESTART 20`

Maximal restarting number

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

10.26.2.66 `#define MAX_STAG 20`

Maximal number of stagnation times

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

10.26.2.67 `#define MIN_CDOF 20`

Minimal number of coarsest variables

Definition at line 241 of file fasp\_const.h.

10.26.2.68 `#define MIN_CRATE 0.9`

Minimal coarsening ratio

Definition at line 242 of file fasp\_const.h.

10.26.2.69 `#define NL_AMLI_CYCLE 4`

Nonlinear AMLI-cycle

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

10.26.2.70 `#define NO_ORDER 0`

Definition of smoothing order.

Natural order smoothing

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

#### 10.26.2.71 `#define OFF 0`

turn off certain parameter

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

#### 10.26.2.72 `#define ON 1`

Definition of switch.

turn on certain parameter

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

#### 10.26.2.73 `#define OPENMP_HOLDS 2000`

Smallest size for OpenMP version

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

#### 10.26.2.74 `#define PAIRWISE 1`

Definition of aggregation types.

pairwise aggregation

Definition at line 168 of file fasp\_const.h.

#### 10.26.2.75 `#define PREC_AMG 2`

with AMG precondition

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

#### 10.26.2.76 `#define PREC_DIAG 1`

with diagonal precondition

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

#### 10.26.2.77 `#define PREC_FMG 3`

with full AMG precondition

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

#### 10.26.2.78 `#define PREC_ILU 4`

with ILU precondition

Definition at line 141 of file fasp\_const.h.

#### 10.26.2.79 `#define PREC_NULL 0`

Definition of preconditioner type for iterative methods.

with no precondition

Definition at line 137 of file fasp\_const.h.

#### 10.26.2.80 `#define PREC_SCHWARZ 5`

with Schwarz preconditioner

Definition at line 142 of file fasp\_const.h.

#### 10.26.2.81 `#define PRINT_ALL 10`

all: all printouts, including files

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

#### 10.26.2.82 `#define PRINT_MIN 1`

quiet: print error, important warnings

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

#### 10.26.2.83 `#define PRINT_MORE 4`

more: print some useful debug info

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

#### 10.26.2.84 `#define PRINT_MOST 8`

most: maximal printouts, no files

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

#### 10.26.2.85 `#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.

#### 10.26.2.86 `#define PRINT_SOME 2`

some: print less important warnings

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



**10.26.2.87 #define SA\_AMG 2**

smoothed aggregation AMG

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

**10.26.2.88 #define SCHWARZ\_BACKWARD 2**

Backward ordering

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

**10.26.2.89 #define SCHWARZ\_FORWARD 1**

Type of Schwarz smoother.

Forward ordering

Definition at line 154 of file fasp\_const.h.

**10.26.2.90 #define SCHWARZ\_SYMMETRIC 3**

Symmetric smoother

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

**10.26.2.91 #define SMALLREAL 1e-20**

A small real number

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

**10.26.2.92 #define SMALLREAL2 1e-40**

An extremely small real number

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

**10.26.2.93 #define SMOOTHER\_CG 4**

CG as a smoother

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

**10.26.2.94 #define SMOOTHER\_GS 2**

Gauss-Seidel smoother

Definition at line 183 of file fasp\_const.h.

**10.26.2.95 #define SMOOTHER\_GSOR 7**

GS + SOR smoother

Definition at line 188 of file fasp\_const.h.

**10.26.2.96 #define SMOOTHER\_JACOBI 1**

Definition of standard smoother types.

Jacobi smoother

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

**10.26.2.97 #define SMOOTHER\_L1DIAG 10**

L1 norm diagonal scaling smoother

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

**10.26.2.98 #define SMOOTHER\_POLY 9**

Polynomial smoother

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

**10.26.2.99 #define SMOOTHER\_SGS 3**

Symmetric Gauss-Seidel smoother

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

**10.26.2.100 #define SMOOTHER\_SGSOR 8**

SGS + SSOR smoother

Definition at line 189 of file fasp\_const.h.

**10.26.2.101 #define SMOOTHER\_SOR 5**

SOR smoother

Definition at line 186 of file fasp\_const.h.

**10.26.2.102 #define SMOOTHER\_SSOR 6**

SSOR smoother

Definition at line 187 of file fasp\_const.h.

**10.26.2.103 #define SOLVER\_AMG 21**

AMG as an iterative solver

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

**10.26.2.104 #define SOLVER\_BiCGstab 2**

Bi-Conjugate Gradient Stabilized

Definition at line 104 of file fasp\_const.h.

**10.26.2.105 #define SOLVER\_CG 1**

Conjugate Gradient

Definition at line 103 of file fasp\_const.h.

**10.26.2.106 #define SOLVER\_DEFAULT 0**

Definition of solver types for iterative methods.

Use default solver in FASP

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

**10.26.2.107 #define SOLVER\_FMG 22**

Full AMG as an solver

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

**10.26.2.108 #define SOLVER\_GCG 7**

Generalized Conjugate Gradient

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

**10.26.2.109 #define SOLVER\_GCR 8**

Generalized Conjugate Residual

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

**10.26.2.110 #define SOLVER\_GMRES 4**

Generalized Minimal Residual

Definition at line 106 of file fasp\_const.h.

10.26.2.111 `#define SOLVER_MinRes 3`

Minimal Residual

Definition at line 105 of file fasp\_const.h.

10.26.2.112 `#define SOLVER_MUMPS 33`

MUMPS Direct Solver

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

10.26.2.113 `#define SOLVER_SBiCGstab 12`

BiCGstab with safety net

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

10.26.2.114 `#define SOLVER_SCG 11`

Conjugate Gradient with safety net

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

10.26.2.115 `#define SOLVER_SGCG 17`

GCG with safety net

Definition at line 118 of file fasp\_const.h.

10.26.2.116 `#define SOLVER_SGMRES 14`

GMRes with safety net

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

10.26.2.117 `#define SOLVER_SMinRes 13`

MinRes with safety net

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

10.26.2.118 `#define SOLVER_SUPERLU 31`

SuperLU Direct Solver

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

10.26.2.119 `#define SOLVER_SVFGMRES 16`

Variable-restart FGMRES with safety net

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

**10.26.2.120 #define SOLVER\_SVGMRES 15**

Variable-restart GMRES with safety net

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

**10.26.2.121 #define SOLVER\_UMFPACK 32**

UMFPack Direct Solver

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

**10.26.2.122 #define SOLVER\_VFGMRES 6**

Variable Restarting Flexible GMRES

Definition at line 108 of file fasp\_const.h.

**10.26.2.123 #define SOLVER\_VGMRES 5**

Variable Restarting GMRES

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

**10.26.2.124 #define STAG\_RATIO 1e-4**

Stagnation tolerance = tol\*STAGRATIO

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

**10.26.2.125 #define STOP\_MOD\_REL\_RES 3**

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

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

**10.26.2.126 #define STOP\_REL\_PRECRES 2**

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

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

**10.26.2.127 #define STOP\_REL\_RES 1**

Definition of iterative solver stopping criteria types.

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

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

**10.26.2.128 #define TRUE 1**

Definition of logic type.

logic TRUE

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

**10.26.2.129 #define UA\_AMG 3**

unsmoothed aggregation AMG

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

**10.26.2.130 #define UNPT -1**

Undetermined points

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

**10.26.2.131 #define USERDEFINED 0**

Type of ordering for smoothers.

User defined order

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

**10.26.2.132 #define V\_CYCLE 1**

Definition of cycle types.

V-cycle

Definition at line 174 of file fasp\_const.h.

**10.26.2.133 #define VMB 2**

VMB aggregation

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

**10.26.2.134 #define W\_CYCLE 2**

W-cycle

Definition at line 175 of file fasp\_const.h.

## **10.27 fmgcycle.c File Reference**

Abstract non-recursive full multigrid cycle.

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

## Functions

- void [fasp\\_solver\\_fmecycle](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
*#include "forts\_ns.h"*

### 10.27.1 Detailed Description

Abstract non-recursive full multigrid cycle.

### 10.27.2 Function Documentation

10.27.2.1 void [fasp\\_solver\\_fmecycle](#) ( [AMG\\_data](#) \* *mgl*, [AMG\\_param](#) \* *param* )

*#include "forts\_ns.h"*

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.

Definition at line 35 of file fmecycle.c.

## 10.28 formats.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` (`dCOOmat *A`, `dCSRmat *B`)  
*Transform a REAL matrix from its IJ format to its CSR format.*
- **SHORT** `fasp_format_dcsr_dcoo` (`dCSRmat *A`, `dCOOmat *B`)  
*Transform a REAL matrix from its CSR format to its IJ format.*
- **SHORT** `fasp_format_dstr_dcsr` (`dSTRmat *A`, `dCSRmat *B`)  
*Transfer a 'dSTRmat' type matrix into a 'dCSRmat' type matrix.*
- `dCSRmat fasp_format_bdcsr_dcsr` (`block_dCSRmat *Ab`)  
*Form the whole dCSRmat A using blocks given in Ab.*
- `dCSRLmat * fasp_format_dcsr_dcsr` (`dCSRmat *A`)  
*Convert a dCSRmat into a dCSRLmat.*
- `dCSRmat fasp_format_dbsr_dcsr` (`dBSRmat *B`)  
*Transfer a 'dBSRmat' type matrix into a dCSRmat.*
- `dBSRmat fasp_format_dcsr_dbsr` (`dCSRmat *A`, `const INT nb`)  
*Transfer a dCSRmat type matrix into a dBSRmat.*
- `dBSRmat fasp_format_dstr_dbsr` (`dSTRmat *B`)  
*Transfer a 'dSTRmat' type matrix to a 'dBSRmat' type matrix.*
- `dCOOmat * fasp_format_dbsr_dcoo` (`dBSRmat *B`)  
*Transfer a 'dBSRmat' type matrix to a 'dCOOmat' type matrix.*

### 10.28.1 Detailed Description

Subroutines for matrix format conversion.

### 10.28.2 Function Documentation

#### 10.28.2.1 `dCSRmat fasp_format_bdcsr_dcsr ( block_dCSRmat * Ab )`

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

##### Parameters

<code>Ab</code>	Pointer to <code>block_dCSRmat</code> matrix
-----------------	--

##### Returns

`dCSRmat` matrix if succeed, NULL if fail

##### Author

Shiquan Zhang

##### Date

08/10/2010

Definition at line 292 of file `formats.c`.

#### 10.28.2.2 `dCOOmat * fasp_format_dbsr_dcoo ( 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 943 of file formats.c.

### 10.28.2.3 [dCSRmat](#) fasp\_format\_dbsr\_dcsr ( [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 495 of file formats.c.

### 10.28.2.4 [SHORT](#) fasp\_format\_dcoo\_dcsr ( [dCOOmat](#) \* $A$ , [dCSRmat](#) \* $B$ )

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

## Parameters

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

## Returns

FASP\_SUCCESS if succeeded; otherwise, error information.

## Author

Xuehai Huang

## Date

08/10/2009

Definition at line 27 of file formats.c.

#### 10.28.2.5 [dBSRmat](#) fasp\_format\_dcsr\_dbsr ( [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 721 of file formats.c.

#### 10.28.2.6 [SHORT](#) fasp\_format\_dcsr\_dcoo ( [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 80 of file formats.c.

#### 10.28.2.7 [dCSRLmat](#) \* fasp\_format\_dcsr( [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 361 of file formats.c.

#### 10.28.2.8 [dBSRmat](#) fasp\_format\_dstr\_dbsr( [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 839 of file formats.c.

### 10.28.2.9 **SHORT** fasp\_format\_dstr\_dcsr ( [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 117 of file formats.c.

## 10.29 **givens.c** File Reference

Givens transformation.

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

**Functions**

- void [fasp\\_aux\\_givens](#) (const [REAL](#) beta, [dCSRmat](#) \*H, [dvector](#) \*y, [REAL](#) \*tmp)  
Perform Givens rotations to compute  $y \mid \beta e_1 - H * y$ .

### 10.29.1 Detailed Description

Givens transformation.

### 10.29.2 Function Documentation

10.29.2.1 void fasp\_aux\_givens ( const REAL beta, dCSRmat \* H, dvector \* y, REAL \* tmp )

Perform Givens rotations to compute  $y \sqrt{|\beta e_1 - H y|}$ .

#### Parameters

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

#### Author

Xuehai Huang

#### Date

10/19/2008

Definition at line 28 of file givens.c.

## 10.30 gmg\_poisson.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 "gmg_util.inl"
```

### Functions

- [INT fasp\\_poisson\\_gmg\\_1D](#) (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\\_gmg\\_2D](#) (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\\_gmg\\_3D](#) (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\\_fgmg\\_1D](#) (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 (Full Multigrid)*

- void fasp\_poisson\_fgmg\_2D (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 (Full Multigrid)*

- void fasp\_poisson\_fgmg\_3D (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 (Full Multigrid)*

- INT fasp\_poisson\_pcg\_gmg\_1D (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\_pcg\_gmg\_2D (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\_pcg\_gmg\_3D (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)*

### 10.30.1 Detailed Description

GMG method as an iterative solver for Poisson Problem.

### 10.30.2 Function Documentation

10.30.2.1 void fasp\_poisson\_fgmg\_1D ( 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 (Full Multigrid)

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

Date

06/07/2013

Definition at line 431 of file gmg\_poisson.c.

10.30.2.2 void fasp\_poisson\_fgmg\_2D ( 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 (Full Multigrid)

## 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

## Date

06/07/2013

Definition at line 524 of file gmg\_poisson.c.

10.30.2.3 void fasp\_poisson\_fgmg\_3D ( 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 (Full Multigrid)

## 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

## Date

06/07/2013

Definition at line 632 of file gmg\_poisson.c.

10.30.2.4 INT fasp\_poisson\_gmg\_1D ( 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

## Date

06/07/2013

Definition at line 36 of file gmg\_poisson.c.

**10.30.2.5** `INT fasp_poisson_gmg_2D ( REAL * u, REAL * b, const INT nx, const INT ny, const INT maxlevel, const REAL rtol, const SHORT prtlvl )`

Solve  $Ax=b$  of Poisson 2D equation by Geometric Multigrid Method.

## Parameters

<i>u</i>	Pointer to the vector of dofs
<i>b</i>	Pointer to the vector of right hand side
<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>maxlevel</i>	Maximum levels of the multigrid
<i>rtol</i>	Relative tolerance to judge convergence
<i>prtlvl</i>	Print level for output

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Ziteng Wang

## Date

06/07/2013

Definition at line 160 of file gmg\_poisson.c.

**10.30.2.6** `INT fasp_poisson_gmg_3D ( 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

## Date

06/07/2013

Definition at line 296 of file gmg\_poisson.c.

**10.30.2.7 INT fasp\_poisson\_pcg\_gmg\_1D ( 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

## Date

06/07/2013

Definition at line 741 of file gmg\_poisson.c.

10.30.2.8 `INT fasp_poisson_pcg_gmg_2D ( 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

## Date

06/07/2013

Definition at line 835 of file gmg\_poisson.c.

**10.30.2.9** `INT fasp_poisson_pcg_gmg_3D ( 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

## Date

06/07/2013

Definition at line 944 of file gmg\_poisson.c.

## 10.31 graphics.c File Reference

Subroutines for graphical output.

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

### Functions

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

### 10.31.1 Detailed Description

Subroutines for graphical output.

### 10.31.2 Function Documentation

#### 10.31.2.1 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 <a href="#">dBSRmat</a> 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 469 of file `graphics.c`.

### 10.31.2.2 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 105 of file `graphics.c`.

### 10.31.2.3 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 628 of file `graphics.c`.

10.31.2.4 `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 44 of file `graphics.c`.

10.31.2.5 `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 172 of file graphics.c.

## 10.32 ilu\_setup\_bsr.c File Reference

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

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

**Functions**

- void **symbfactor\_** (const [INT](#) \*n, [INT](#) \*colind, [INT](#) \*rowptr, const [INT](#) \*levfill, const [INT](#) \*nzmax, [INT](#) \*nzlu, [INT](#) \*ijlu, [INT](#) \*uptr, [INT](#) \*ierr)
- [SHORT fasp\\_ilu\\_dbsr\\_setup](#) ([dBSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)  
*Get ILU decoposition of a BSR matrix A.*

### 10.32.1 Detailed Description

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

### 10.32.2 Function Documentation

#### 10.32.2.1 [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 45 of file ilu\_setup\_bsr.c.

## 10.33 ilu\_setup\_csr.c File Reference

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

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

### Functions

- void [iluk\\_](#) (const [INT](#) \*n, [REAL](#) \*a, [INT](#) \*ja, [INT](#) \*ia, [INT](#) \*lfil, [REAL](#) \*alu, [INT](#) \*jlu, [INT](#) \*iwk, [INT](#) \*ierr, [INT](#) \*nzlu)
- void [ilut\\_](#) (const [INT](#) \*n, [REAL](#) \*a, [INT](#) \*ja, [INT](#) \*ia, [INT](#) \*lfil, const [REAL](#) \*droptol, [REAL](#) \*alu, [INT](#) \*jlu, [INT](#) \*iwk, [INT](#) \*ierr, [INT](#) \*nz)
- void [ilutp\\_](#) (const [INT](#) \*n, [REAL](#) \*a, [INT](#) \*ja, [INT](#) \*ia, [INT](#) \*lfil, const [REAL](#) \*droptol, const [REAL](#) \*permtol, const [INT](#) \*mbloc, [REAL](#) \*alu, [INT](#) \*jlu, [INT](#) \*iwk, [INT](#) \*ierr, [INT](#) \*nz)
- [SHORT fasp\\_ilu\\_dcsr\\_setup](#) ([dCSRmat](#) \*A, [ILU\\_data](#) \*iludata, [ILU\\_param](#) \*iluparam)

*Get ILU decomposition of a CSR matrix A.*

### 10.33.1 Detailed Description

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

### 10.33.2 Function Documentation

#### 10.33.2.1 [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 50 of file ilu\_setup\_csr.c.

## 10.34 ilu\_setup\_str.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) decoposition of a structured matrix A.*

### 10.34.1 Detailed Description

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

### 10.34.2 Function Documentation

10.34.2.1 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 28 of file `ilu_setup_str.c`.

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

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

**Parameters**

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

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

11/08/2010

**Note**

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

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

Definition at line 319 of file `ilu_setup_str.c`.

## 10.35 `init.c` File Reference

Initialize important data structures.

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

## Functions

- void `fasp_precond_data_null` (`precond_data` \*pcdata)  
*Initialize `precond_data`.*
- `AMG_data` \* `fasp_amg_data_create` (`SHORT` max\_levels)  
*Create and initialize `AMG_data` for classical and SA AMG.*
- `AMG_data_bsr` \* `fasp_amg_data_bsr_create` (`SHORT` max\_levels)  
*Create and initialize `AMG_data` data sturcture for AMG/SAMG (BSR format)*
- void `fasp_ilu_data_alloc` (const `INT` iwk, const `INT` nwork, `ILU_data` \*iludata)  
*Allocate workspace for ILU factorization.*
- void `fasp_Schwarz_data_free` (`Schwarz_data` \*Schwarz)  
*Free `Schwarz_data` data memeory space.*
- void `fasp_amg_data_free` (`AMG_data` \*mgl, `AMG_param` \*param)  
*Free `AMG_data` data memeory space.*
- void `fasp_amg_data_bsr_free` (`AMG_data_bsr` \*mgl)  
*Free `AMG_data_bsr` data memeory space.*
- void `fasp_ilu_data_free` (`ILU_data` \*ILUdata)  
*Create `ILU_data` sturcture.*
- void `fasp_ilu_data_null` (`ILU_data` \*ILUdata)  
*Initialize ILU data.*
- void `fasp_precond_null` (`precond` \*pcdata)  
*Initialize `precond` data.*

### 10.35.1 Detailed Description

Initialize important data structures.

#### Note

Every structures should be initialized before usage.

### 10.35.2 Function Documentation

#### 10.35.2.1 `AMG_data_bsr` \* `fasp_amg_data_bsr_create` ( `SHORT` max\_levels )

Create and initialize `AMG_data` data sturcture for AMG/SAMG (BSR format)

#### Parameters

<code>max_levels</code>	Max number of levels allowed
-------------------------	------------------------------

#### Returns

Pointer to the `AMG_data` data structure

#### Author

Xiaozhe Hu

#### Date

08/07/2011

Definition at line 86 of file init.c.

### 10.35.2.2 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 249 of file init.c.

### 10.35.2.3 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 56 of file init.c.

### 10.35.2.4 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!

Definition at line 184 of file init.c.

10.35.2.5 void fasp\_ilu\_data\_alloc ( 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 118 of file init.c.

10.35.2.6 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 293 of file init.c.

10.35.2.7 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 314 of file init.c.

10.35.2.8 void fasp\_precond\_data\_null ( **precond\_data** \* *pcdata* )

Initialize [precond\\_data](#).

## Parameters

<i>pcdata</i>	Preconditioning data structure
---------------	--------------------------------

## Author

Chensong Zhang

## Date

2010/03/23

Definition at line 25 of file init.c.

10.35.2.9 void fasp\_precond\_null ( **precond** \* *pcdata* )

Initialize precondition data.

## Parameters

<i>pcdata</i>	Pointer to precondition
---------------	-------------------------

## Author

Chensong Zhang

## Date

2010/03/23

Definition at line 330 of file init.c.

10.35.2.10 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 147 of file init.c.

## 10.36 input.c File Reference

Read 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 \*filenm, [input\\_param](#) \*inparam)  
*Read input parameters from disk file.*

### 10.36.1 Detailed Description

Read input parameters.

### 10.36.2 Function Documentation

#### 10.36.2.1 [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 25 of file input.c.

10.36.2.2 `void fasp_param_input ( const char * filenm, input_param * inparam )`

Read input parameters from disk file.

**Parameters**

<i>filenm</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 102 of file input.c.

**10.37 interface\_mumps.c File Reference**

Interface to MUMPS direct solvers.

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

**Macros**

- `#define ICNTL(l) icntl[(l)-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.



### 10.37.1 Detailed Description

Interface to MUMPS direct solvers.

Reference for MUMPS: <http://mumps.enseeiht.fr/>

### 10.37.2 Macro Definition Documentation

#### 10.37.2.1 `#define ICNTL( I ) icntl[(I)-1]`

macro s.t. indices match documentation

Definition at line 17 of file interface\_mumps.c.

### 10.37.3 Function Documentation

#### 10.37.3.1 `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 39 of file interface\_mumps.c.

#### 10.37.3.2 `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 169 of file interface\_mumps.c.

## 10.38 interface\_samg.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.*

### 10.38.1 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)!

### 10.38.2 Function Documentation

#### 10.38.2.1 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 59 of file interface\_samg.c.

## 10.38.2.2 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 30 of file interface\_samg.c.

## 10.39 interface\_superlu.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.*

## 10.39.1 Detailed Description

Interface to SuperLU direct solvers.

Reference for SuperLU: <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>

## 10.39.2 Function Documentation

## 10.39.2.1 int fasp\_solver\_superlu ( dCSRmat \* ptrA, dvector \* b, dvector \* u, const SHORT prtlvl )

Solve  $Au=b$  by SuperLU.

**Parameters**

<i>ptrA</i>	Pointer to a <a href="#">dCSRmat</a> matrix
<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 40 of file interface\_superlu.c.

**10.40 interface\_umfpack.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.*

**10.40.1 Detailed Description**

Interface to UMFPACK direct solvers.

Reference for SuiteSparse: <http://faculty.cse.tamu.edu/davis/suitesparse.html>

**10.40.2 Function Documentation**

**10.40.2.1** [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 37 of file interface\_umfpack.c.

## 10.41 interpolation.c File Reference

Interpolation operators for 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\\_interp1](#) ([dCSRmat](#) \*A, [ivector](#) \*vertices, [dCSRmat](#) \*P, [AMG\\_param](#) \*param, [iCSRmat](#) \*S, [INT](#) \*icor\_ysk)  
*Generate interpolation operator P.*
- void [fasp\\_amg\\_interp\\_trunc](#) ([dCSRmat](#) \*P, [AMG\\_param](#) \*param)  
*Truncation step for prolongation operators.*

### 10.41.1 Detailed Description

Interpolation operators for AMG.

**Note**

Ref 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.

### 10.41.2 Function Documentation

10.41.2.1 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> : the 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 48 of file interpolation.c.

10.41.2.2 void fasp\_amg\_interp1 ( dCSRmat \* *A*, ivector \* *vertices*, dCSRmat \* *P*, AMG\_param \* *param*, iCSRmat \* *S*, INT \* *icor\_ysk* )

Generate interpolation operator P.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the 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
<i>icor_ysk</i>	Indices of coarse nodes in fine grid

## Returns

FASP\_SUCCESS or error message

## Author

Chunsheng Feng, Xiaoqiang Yue

## Date

03/01/2011

Modified by Chensong Zhang on 05/14/2013: reconstruct the code

Definition at line 105 of file interpolation.c.

10.41.2.3 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 159 of file interpolation.c.

## 10.42 interpolation\_em.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.*

### 10.42.1 Detailed Description

Interpolation operators for AMG based on energy-min.

## Note

Ref J. Xu and L. Zikatanov "On An Energy Minimizing Basis in Algebraic Multigrid Methods" Computing and visualization in sciences, 2003

### 10.42.2 Function Documentation

10.42.2.1 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 49 of file interpolation\_em.c.

## 10.43 io.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` (INT n, `dvector` \*u)
- Print first n entries of a vector of REAL type.*

  - void `fasp_ivec_print` (INT n, `ivector` \*u)
- Print first n entries of a vector of INT type.*

  - void `fasp_dcsr_print` (`dCSRmat` \*A)
- Print out a dCSRmat matrix in coordinate format.*

  - void `fasp_dcoo_print` (`dCOOmat` \*A)
- Print out a dCOOmat matrix in coordinate format.*

  - void `fasp_dbsr_print` (`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` (`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, 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, [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

### 10.43.1 Detailed Description

Matrix/vector input/output subroutines.

#### Note

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

### 10.43.2 Function Documentation

#### 10.43.2.1 void [fasp\\_dbsr\\_print](#) ( [dBSRmat](#) \* A )

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

#### Parameters

<a href="#">A</a>	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 1444 of file io.c.

#### 10.43.2.2 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 691 of file io.c.

**10.43.2.3** 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 1202 of file io.c.

**10.43.2.4** 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 <a href="#">dBSRmat</a> matrix <i>A</i>

## Author

Chunsheng Feng

## Date

11/14/2013

Modified by Chensong Zhang on 06/14/2014: Fix index problem.

Definition at line 1481 of file io.c.

10.43.2.5 `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 369 of file io.c.

10.43.2.6 `void fasp_dcoo_print ( dCOOmat * A )`

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

## Parameters

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

**Author**

Ziteng Wang

**Date**

12/24/2012

Definition at line 1423 of file io.c.

**10.43.2.7** 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 318 of file io.c.

**10.43.2.8** 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 420 of file io.c.

10.43.2.9 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 1102 of file io.c.

10.43.2.10 void fasp\_dcsr\_print ( 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 1401 of file io.c.

10.43.2.11 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 257 of file io.c.

10.43.2.12 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 A

## Author

Chunsheng Feng

## Date

11/14/2013

Definition at line 1531 of file io.c.

10.43.2.13 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 86 of file `io.c`.

10.43.2.14 `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 953 of file io.c.

10.43.2.15 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 178 of file io.c.

10.43.2.16 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 1031 of file io.c.

10.43.2.17 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 dCS↵  
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 472 of file io.c.

10.43.2.18 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 534 of file io.c.

10.43.2.19 void fasp\_dstr\_print ( dSTRmat \* *A* )

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

## Parameters

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

## Author

Ziteng Wang

## Date

12/24/2012

Definition at line 1570 of file io.c.

10.43.2.20 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 611 of file io.c.

**10.43.2.21** 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 1142 of file io.c.

**10.43.2.22** void fasp\_dvec\_print ( 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 1362 of file io.c.

10.43.2.23 void fasp\_dvec\_read ( const char \* *filename*, dvector \* *b* )

Read b from a disk file in array format.

## Parameters

<i>filename</i>	File name for vector b
<i>b</i>	Pointer to the dvector b (output)

## Note

File Format:

- nrow
- val\_j, j=0:nrow-1

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 810 of file io.c.

10.43.2.24 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 1257 of file io.c.

10.43.2.25 void fasp\_dvecind\_read ( const char \* *filename*, dvector \* *b* )

Read *b* from matrix disk file.

Parameters

<i>filename</i>	File name for vector <i>b</i>
<i>b</i>	Pointer to the dvector <i>b</i> (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 760 of file io.c.

10.43.2.26 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 1293 of file io.c.

10.43.2.27 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 2061 of file io.c.

10.43.2.28 void fasp\_ivec\_print ( INT *n*, ivector \* *u* )

Print first *n* entries of a vector of INT type.

## Parameters

<i>n</i>	An interger (if <i>n</i> =0, then print all entries)
<i>u</i>	Pointer to an ivector

## Author

Chensong Zhang

## Date

03/29/2009

Definition at line 1382 of file io.c.

10.43.2.29 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 <i>b</i>
<i>b</i>	Pointer to the dvector <i>b</i> (output)

## Note

File Format:

- *nrow*
- *val\_j*, *j*=0:*nrow*-1



**Author**

Xuehai Huang

**Date**

03/29/2009

Definition at line 902 of file io.c.

**10.43.2.30** 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 1328 of file io.c.

**10.43.2.31** 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 862 of file io.c.

10.43.2.32 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 1604 of file io.c.

10.43.2.33 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 1709 of file io.c.

10.43.2.34 `fasp_matrix_write ( const char * filemat, void * A, 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 1783 of file io.c.

10.43.2.35 `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 1876 of file io.c.

10.43.2.36 `fasp_vector_write ( const char * filerhs, void * b, 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 1973 of file io.c.

### 10.43.3 Variable Documentation

#### 10.43.3.1 INT dlength

Length of REAL in byte

Definition at line 14 of file io.c.

#### 10.43.3.2 INT ilength

Length of INT in byte

Definition at line 13 of file io.c.

## 10.44 itsolver\_bcsr.c File Reference

Iterative solvers for [block\\_dCSRmat](#) matrices.

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

### Functions

- [INT fasp\\_solver\\_bdcsr\\_itsolver](#) ([block\\_dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*
- [INT fasp\\_solver\\_bdcsr\\_krylov](#) ([block\\_dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [itsolver\\_param](#) \*itparam)  
*Solve  $Ax = b$  by standard Krylov methods.*

- `INT fasp_solver_bdcsr_krylov_block_3` (`block_dCSRmat *A`, `dvector *b`, `dvector *x`, `itsolver_param *itparam`, `AMG_param *amgparam`, `dCSRmat *A_diag`)  
Solve  $Ax = b$  by standard Krylov methods.
- `INT fasp_solver_bdcsr_krylov_block_4` (`block_dCSRmat *A`, `dvector *b`, `dvector *x`, `itsolver_param *itparam`, `AMG_param *amgparam`, `dCSRmat *A_diag`)  
Solve  $Ax = b$  by standard Krylov methods.
- `INT fasp_solver_bdcsr_krylov_sweeping` (`block_dCSRmat *A`, `dvector *b`, `dvector *x`, `itsolver_param *itparam`, `INT NumLayers`, `block_dCSRmat *Ai`, `dCSRmat *local_A`, `ivector *local_index`)  
Solve  $Ax = b$  by standard Krylov methods.

### 10.44.1 Detailed Description

Iterative solvers for `block_dCSRmat` matrices.

### 10.44.2 Function Documentation

10.44.2.1 `INT fasp_solver_bdcsr_itsolver ( block_dCSRmat * A, dvector * b, dvector * x, precondition * pc, itsolver_param * itparam )`

Solve  $Ax = b$  by standard Krylov methods.

#### Parameters

<code>A</code>	Pointer to the coeff matrix in <code>block_dCSRmat</code> format
<code>b</code>	Pointer to the right hand side in dvector format
<code>x</code>	Pointer to the approx solution in dvector format
<code>pc</code>	Pointer to the preconditioning action
<code>itparam</code>	Pointer to parameters for iterative solvers

#### Returns

Iteration number if converges; ERROR otherwise.

#### Author

Chensong Zhang

#### Date

11/25/2010

Definition at line 36 of file `itsolver_bcsr.c`.

10.44.2.2 `INT fasp_solver_bdcsr_krylov ( block_dCSRmat * 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="#">block_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

Xiaozhe Hu

## Date

07/18/2010

Definition at line 123 of file itsolver\_bcsr.c.

**10.44.2.3** `INT fasp_solver_bdcsl_krylov_block_3 ( block_dCSRmat * 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="#">block_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 solvers
<i>A_diag</i>	Digonal blocks of A

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

07/10/2014

## Note

only works for 3by3 block [dCSRmat](#) problems!! – Xiaozhe Hu

Definition at line 177 of file itsolver\_bcsr.c.

**10.44.2.4** `INT fasp_solver_bdcsl_krylov_block_4 ( block_dCSRmat * 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="#">block_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 solvers
<i>A_diag</i>	Digonal blocks of A

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Xiaozhe Hu

## Date

07/06/2014

## Note

only works for 4 by 4 block [dCSRmat](#) problems!! – Xiaozhe Hu

Definition at line 383 of file itsolver\_bcsr.c.

10.44.2.5 **INT fasp\_solver\_bdcsl\_krylov\_sweeping ( [block\\_dCSRmat](#) \* *A*, [dvector](#) \* *b*, [dvector](#) \* *x*, [itsolver\\_param](#) \* *itparam*, INT *NumLayers*, [block\\_dCSRmat](#) \* *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="#">block_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>NumLayers</i>	Number of layers used for sweeping preconditioner
<i>Ai</i>	Pointer to the coeff matrix for the preconditioner in <a href="#">block_dCSRmat</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 509 of file itsolver\_bcsr.c.



## 10.45 itsolver\_bsr.c File Reference

Iterative solvers for [dBSRmat](#) matrices.

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

### 10.45.1 Detailed Description

Iterative solvers for [dBSRmat](#) matrices.

### 10.45.2 Function Documentation

**10.45.2.1** [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.

Parameters

A	Pointer to the coeff matrix in <a href="#">dBSRmat</a> format
b	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

Definition at line 37 of file itsolver\_bsr.c.

**10.45.2.2 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 125 of file itsolver\_bsr.c.

**10.45.2.3 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 347 of file itsolver\_bsr.c.

10.45.2.4 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 488 of file itsolver\_bsr.c.

10.45.2.5 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 176 of file itsolver\_bsr.c.

10.45.2.6 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 280 of file itsolver\_bsr.c.

10.45.2.7 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 647 of file itsolver\_bsr.c.

## 10.46 itsolver\_csr.c File Reference

Iterative solvers for [dCSRmat](#) matrices.

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

### 10.46.1 Detailed Description

Iterative solvers for `dCSRmat` matrices.

### 10.46.2 Function Documentation

- 10.46.2.1 `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.

#### Parameters

<i>A</i>	Pointer to the coeff matrix in <code>dCSRmat</code> format
<i>b</i>	Pointer to the right hand side in <code>dvector</code> format
<i>x</i>	Pointer to the approx solution in <code>dvector</code> 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.

Definition at line 39 of file `itsolver_csr.c`.

- 10.46.2.2 `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 143 of file itsolver\_csr.c.

**10.46.2.3** `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 338 of file itsolver\_csr.c.

**10.46.2.4** `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 611 of file itsolver\_csr.c.

10.46.2.5 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 193 of file itsolver\_csr.c.

10.46.2.6 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 443 of file itsolver\_csr.c.

**10.46.2.7** `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 527 of file itsolver\_csr.c.

10.46.2.8 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 257 of file itsolver\_csr.c.

## 10.47 itsolver\_mf.c File Reference

Iterative solvers using matrix-free spmv operations.

```
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp_block.h"
#include "itsolver_util.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\\_itsolver\\_init](#) ([INT](#) matrix\_format, [mxv\\_matfree](#) \*mf, void \*A)  
*Initialize itsolvers.*

### 10.47.1 Detailed Description

Iterative solvers using matrix-free spmv operations.

## 10.47.2 Function Documentation

10.47.2.1 `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> matrix-free 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 50 of file itsolver\_mf.c.

**10.47.2.2** void fasp\_solver\_itsolver\_init ( INT *matrix\_format*, mxv\_matfree \* *mf*, void \* *A* )

Initialize itsolvers.

## Parameters

<i>matrix_format</i>	matrix format
<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> matrix-free spmv operation
<i>A</i>	void pointer to matrix

## Author

Feiteng Huang

## Date

09/18/2012

Modified by Chensong Zhang on 05/10/2013: Change interface of mat-free mv

Definition at line 197 of file itsolver\_mf.c.

**10.47.2.3** 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> matrix-free 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 150 of file itsolver\_mf.c.

**10.48 itsolver\_str.c File Reference**

Iterative solvers for [dSTRmat](#) matrices.

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

**10.48.1 Detailed Description**

Iterative solvers for [dSTRmat](#) matrices.

## 10.48.2 Function Documentation

**10.48.2.1** `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

Definition at line 34 of file itsolver\_str.c.

**10.48.2.2** `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 117 of file itsolver\_str.c.

10.48.2.3 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 324 of file itsolver\_str.c.

#### 10.48.2.4 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 165 of file itsolver\_str.c.

#### 10.48.2.5 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 231 of file itsolver\_str.c.

## 10.49 lu.c File Reference

LU decomposition and direct solver for small dense matrices.

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

## Functions

- [SHORT fasp\\_smat\\_lu\\_decomp](#) ([REAL](#) \*A, [INT](#) pivot[], const [INT](#) n)  
*LU decomposition of A usind Doolittle's method.*
- [SHORT fasp\\_smat\\_lu\\_solve](#) ([REAL](#) \*A, [REAL](#) b[], [INT](#) pivot[], [REAL](#) x[], const [INT](#) n)  
*Solving  $Ax=b$  using LU decomposition.*

### 10.49.1 Detailed Description

LU decomposition and direct solver for small dense matrices.

### 10.49.2 Function Documentation

#### 10.49.2.1 SHORT fasp\_smat\_lu\_decomp ( [REAL](#) \* A, [INT](#) pivot[], const [INT](#) n )

LU decomposition of A usind 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 46 of file lu.c.

10.49.2.2 **SHORT** fasp\_smat\_lu\_solve ( **REAL** \* *A*, **REAL** *b*[], **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
<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 117 of file lu.c.

## 10.50 memory.c File Reference

Memory allocation and deallocation subroutines.

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

### Functions

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

### 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.*

#### 10.50.1 Detailed Description

Memory allocation and deallocation subroutines.

## 10.50.2 Function Documentation

### 10.50.2.1 void \* fasp\_mem\_calloc ( LONGLONG size, 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 60 of file memory.c.

### 10.50.2.2 SHORT fasp\_mem\_check ( void \* ptr, const char \* message, INT 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 197 of file memory.c.

### 10.50.2.3 SHORT fasp\_mem\_dcsr\_check ( dCSRmat \* A )

Check whether a [dCSRmat](#) A has successfully allocated memory.

## Parameters

<i>A</i>	Pointer to be cheked
----------	----------------------

## Returns

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

## Author

Xiaozhe Hu

## Date

11/27/09

Definition at line 248 of file memory.c.

#### 10.50.2.4 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 150 of file memory.c.

#### 10.50.2.5 SHORT fasp\_mem\_iludata\_check ( ILU\_data \* *iludata* )

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

## Parameters

<i>iludata</i>	Pointer to be cheked
----------------	----------------------

## Returns

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

**Author**

Xiaozhe Hu, Chensong Zhang

**Date**

11/27/09

Definition at line 222 of file memory.c.

**10.50.2.6 void \* fasp\_mem\_realloc ( void \* *oldmem*, **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 110 of file memory.c.

**10.50.2.7 void fasp\_mem\_usage ( )**

Show total allocated memory currently.

**Author**

Chensong Zhang

**Date**

2010/08/12

Definition at line 175 of file memory.c.

**10.50.3 Variable Documentation****10.50.3.1 unsigned INT total\_alloc\_count = 0**

Total allocated memory amount.

total allocation times

Definition at line 35 of file memory.c.

### 10.50.3.2 unsigned INT total\_alloc\_mem = 0

total allocated memory

Definition at line 34 of file memory.c.

## 10.51 message.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](#) ([AMG\\_data](#) \*mgl, const [SHORT](#) ptrlvl)  
*Print complexities of AMG method.*
- void [print\\_amgcomplexity\\_bsr](#) ([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.*

### 10.51.1 Detailed Description

Output some useful messages.

#### Note

These routines are meant for internal use only.

### 10.51.2 Function Documentation

#### 10.51.2.1 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 199 of file message.c.

10.51.2.2 void void print\_amgcomplexity ( **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 message.c.

10.51.2.3 void void print\_amgcomplexity\_bsr ( **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 122 of file message.c.

10.51.2.4 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 165 of file message.c.

10.51.2.5 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 message.c.

10.51.2.6 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 182 of file message.c.

## 10.52 mgcycle.c File Reference

Abstract multigrid cycle – non-recursive version.

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

### Functions

- void [fasp\\_solver\\_mgcycle](#) ([AMG\\_data](#) \*mgl, [AMG\\_param](#) \*param)  
    *#include "forts\_ns.h"*
- void [fasp\\_solver\\_mgcycle\\_bsr](#) ([AMG\\_data\\_bsr](#) \*mgl, [AMG\\_param](#) \*param)  
    *Solve  $Ax=b$  with non-recursive multigrid cycle.*

### 10.52.1 Detailed Description

Abstract multigrid cycle – non-recursive version.

### 10.52.2 Function Documentation

#### 10.52.2.1 void fasp\_solver\_mgcycle ( [AMG\\_data](#) \* *mgl*, [AMG\\_param](#) \* *param* )

#include "forts\_ns.h"

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 41 of file mgcycle.c.

10.52.2.2 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 257 of file mgcycle.c.

## 10.53 mgrecur.c File Reference

Abstract multigrid cycle – recursive version.

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

### Functions

- void [fasp\\_solver\\_mgrecur](#) (AMG\_data \*mgl, AMG\_param \*param, INT level)  
*Solve  $Ax=b$  with recursive multigrid K-cycle.*

#### 10.53.1 Detailed Description

Abstract multigrid cycle – recursive version.

Note

Not used any more. Will be removed! –Chensong

#### 10.53.2 Function Documentation

10.53.2.1 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 33 of file mgrecur.c.

## 10.54 ordering.c File Reference

Subroutines for ordering, merging, removing duplicated integers.

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

## Functions

- [INT fasp\\_BinarySearch](#) (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\\_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.*

### 10.54.1 Detailed Description

Subroutines for ordering, merging, removing duplicated integers.

### 10.54.2 Function Documentation

#### 10.54.2.1 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 239 of file ordering.c.

#### 10.54.2.2 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 320 of file ordering.c.

10.54.2.3 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 201 of file ordering.c.

10.54.2.4 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 279 of file ordering.c.

10.54.2.5 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 108 of file ordering.c.

10.54.2.6 `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 170 of file ordering.c.

10.54.2.7 `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 75 of file ordering.c.

**10.54.2.8 INT fasp\_BinarySearch ( 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 30 of file ordering.c.

**10.54.2.9 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 356 of file ordering.c.

10.54.2.10 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 405 of file ordering.c.

## 10.55 parameters.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](#) (int argc, const char \*argv[], [input\\_param](#) \*iniparam)  
*Read input from command-line arguments.*

- void `fasp_param_init` (`input_param` \*iniparam, `itsolver_param` \*itsparam, `AMG_param` \*amgparam, `ILU_param` \*iluparam, `Schwarz_param` \*schparam)  
*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` \*schparam)  
*Initialize Schwarz parameters.*
- void `fasp_param_amg_set` (`AMG_param` \*param, `input_param` \*iniparam)  
*Set AMG\_param from INPUT.*
- void `fasp_param_ilu_set` (`ILU_param` \*iluparam, `input_param` \*iniparam)  
*Set ILU\_param with INPUT.*
- void `fasp_param_Schwarz_set` (`Schwarz_param` \*schparam, `input_param` \*iniparam)  
*Set Schwarz\_param with INPUT.*
- void `fasp_param_solver_set` (`itsolver_param` \*itsparam, `input_param` \*iniparam)  
*Set itsolver\_param with INPUT.*
- void `fasp_param_amg_to_prec` (`precond_data` \*pcdata, `AMG_param` \*amgparam)  
*Set precondition\_data with AMG\_param.*
- void `fasp_param_prec_to_amg` (`AMG_param` \*amgparam, `precond_data` \*pcdata)  
*Set AMG\_param with precondition\_data.*
- void `fasp_param_amg_to_prec_bsr` (`precond_data_bsr` \*pcdata, `AMG_param` \*amgparam)  
*Set precondition\_data\_bsr with AMG\_param.*
- void `fasp_param_prec_to_amg_bsr` (`AMG_param` \*amgparam, `precond_data_bsr` \*pcdata)  
*Set AMG\_param with precondition\_data.*
- void `fasp_param_amg_print` (`AMG_param` \*param)  
*Print out AMG parameters.*
- void `fasp_param_ilu_print` (`ILU_param` \*param)  
*Print out ILU parameters.*
- void `fasp_param_Schwarz_print` (`Schwarz_param` \*param)  
*Print out Schwarz parameters.*
- void `fasp_param_solver_print` (`itsolver_param` \*param)  
*Print out itsolver parameters.*

### 10.55.1 Detailed Description

Initialize, set, or print input data and parameters.

### 10.55.2 Function Documentation

#### 10.55.2.1 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 390 of file parameters.c.

10.55.2.2 void fasp\_param\_amg\_print ( AMG\_param \* *param* )

Print out AMG parameters.

## Parameters

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

## Author

Chensong Zhang

## Date

2010/03/22

Definition at line 797 of file parameters.c.

10.55.2.3 void fasp\_param\_amg\_set ( AMG\_param \* *param*, 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 518 of file parameters.c.

10.55.2.4 void fasp\_param\_amg\_to\_prec ( precondition\_data \* *pcdata*, 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 666 of file parameters.c.

10.55.2.5 void fasp\_param\_amg\_to\_prec\_bsr ( precondition\_data\_bsr \* *pcdata*, 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 733 of file parameters.c.

10.55.2.6 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 476 of file parameters.c.

10.55.2.7 void fasp\_param\_ilu\_print ( ILU\_param \* *param* )

Print out ILU parameters.

## Parameters

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

## Author

Chensong Zhang

## Date

2011/12/20

Definition at line 898 of file parameters.c.

10.55.2.8 void fasp\_param\_ilu\_set ( ILU\_param \* *iluparam*, 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 593 of file parameters.c.

10.55.2.9 void fasp\_param\_init ( input\_param \* *iniparam*, itsolver\_param \* *itsparam*, AMG\_param \* *amgparam*, ILU\_param \* *iluparam*, Schwarz\_param \* *schparam* )

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>schparam</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 270 of file parameters.c.

10.55.2.10 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 310 of file parameters.c.

10.55.2.11 void fasp\_param\_prec\_to\_amg ( AMG\_param \* *amgparam*, precondition\_data \* *pcdata* )

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 701 of file parameters.c.

10.55.2.12 void fasp\_param\_prec\_to\_amg\_bsr ( AMG\_param \* *amgparam*, precondition\_data\_bsr \* *pcdata* )

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

Parameters

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

Author

Xiaozhe Hu

Date

02/06/2012

Definition at line 767 of file parameters.c.



10.55.2.13 void fasp\_param\_Schwarz\_init ( Schwarz\_param \* *schparam* )

Initialize Schwarz parameters.

## Parameters

<i>schparam</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 498 of file parameters.c.

10.55.2.14 void fasp\_param\_Schwarz\_print ( Schwarz\_param \* *param* )

Print out Schwarz parameters.

## Parameters

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

## Author

Xiaozhe Hu

## Date

05/22/2012

Definition at line 928 of file parameters.c.

10.55.2.15 void fasp\_param\_Schwarz\_set ( Schwarz\_param \* *schparam*, input\_param \* *iniparam* )

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

## Parameters

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

## Author

Xiaozhe Hu

## Date

05/22/2012

Definition at line 615 of file parameters.c.

10.55.2.16 void fasp\_param\_set ( 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 27 of file parameters.c.

10.55.2.17 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 455 of file parameters.c.

10.55.2.18 void fasp\_param\_solver\_print ( itsolver\_param \* *param* )

Print out itsolver parameters.

## Parameters

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

## Author

Chensong Zhang

## Date

2011/12/20

Definition at line 957 of file parameters.c.

10.55.2.19 void fasp\_param\_solver\_set ( itsolver\_param \* *itsparam*, 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 636 of file parameters.c.

## 10.56 pbcgs.c File Reference

Krylov subspace methods – Preconditioned BiCGstab.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "itsolver_util.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](#) prtIvl)  
*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](#) prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*
- [INT fasp\\_solver\\_bdcsr\\_pbcgs](#) ([block\\_dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop\_type, const [SHORT](#) prtIvl)  
*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](#) prtIvl)  
*Preconditioned BiCGstab method for solving  $Au=b$ .*

### 10.56.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab.

Abstract algorithm

PBICGStab method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$ Note: We generate a series of  $\{p_k\}$  such that  $V_k=\text{span}\{p_1,\dots,p_k\}$ .Step 0. Given  $A, b, x_0, M$ Step 1. Compute residual  $r_0 = b - A*x_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1} * r_0$ ,  $p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:MaxIt$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha * p_k$ ;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha * (A * p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;
- print the result of  $k$ -th iteration; END FOR

Convergence check:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check:

- IF  $\text{norm}(\alpha * p_k) / \text{norm}(x_{k+1}) < \text{tol\_stag}$ 
  1. compute  $r = b - A * x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Stag\_Check ) restart;
- END IF

Residual check:

- IF  $\text{norm}(r_{k+1}) / \text{norm}(b) < \text{tol}$ 
  1. compute the real residual  $r = b - A * x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Res\_Check ) restart;
- END IF

Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [spbcgs.c](#) for a safer version

## 10.56.2 Function Documentation

10.56.2.1 INT fasp\_solver\_bdcslr\_pbcgs ( block\_dCSRmat \* 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 the coefficient matrix
<i>b</i>	Pointer to the dvector of right hand side
<i>u</i>	Pointer to the dvector of DOFs
<i>pc</i>	Pointer to precondition: the 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 774 of file pbcgs.c.

10.56.2.2 **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  $Au=b$ .

## Parameters

<i>A</i>	Pointer to the coefficient matrix
<i>b</i>	Pointer to the dvector of right hand side
<i>u</i>	Pointer to the dvector of DOFs
<i>pc</i>	Pointer to precondition: the 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 431 of file pbcgs.c.

**10.56.2.3** `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 the coefficient matrix
<i>b</i>	Pointer to the dvector of right hand side
<i>u</i>	Pointer to the dvector of DOFs
<i>pc</i>	Pointer to precondition: the 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 88 of file pbcgs.c.

**10.56.2.4** `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 the coefficient matrix
----------	-----------------------------------

<i>b</i>	Pointer to the dvector of right hand side
<i>u</i>	Pointer to the dvector of DOFs
<i>pc</i>	Pointer to precondition: the 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 1117 of file pbcgs.c.

## 10.57 pbcgs\_mf.c File Reference

Krylov subspace methods – Preconditioned BiCGstab (matrix free)

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

### Functions

- [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  $Au=b$ .*

#### 10.57.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab (matrix free)

Abstract algorithm of Krylov method

Krylov method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$ , where  $x_k$  is the optimal solution in Krylov space

$V_k = \text{span}\{r_0, A*r_0, A^2*r_0, \dots, A^{k-1}*r_0\}$ ,

under some inner product.



For the implementation, we generate a series of  $\{p_k\}$  such that  $V_k = \text{span}\{p_1, \dots, p_k\}$ . Details:

Step 0. Given  $A, b, x_0, M$

Step 1. Compute residual  $r_0 = b - A * x_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1} * r_0, p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0 : \text{MaxIt}$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha * p_k$ ;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha * (A * p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;
- print the result of  $k$ -th iteration; END FOR

Convergence check is:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check is like following:

- 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 is like following:

- 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

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

### 10.57.2 Function Documentation

10.57.2.1 **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> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 91 of file pbcgs\_mf.c.

## 10.58 pcg.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "itsolver_util.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\\_bdcsr\\_pcg](#) ([block\\_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\\_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  $Au=b$ .*

### 10.58.1 Detailed Description

Krylov subspace methods – Preconditioned conjugate gradient.

Abstract algorithm

PCG method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$

Step 0. Given  $A, b, x_0, M$

Step 1. Compute residual  $r_0 = b - A*x_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1}*r_0, p_0=z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:MaxIt$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha*p_k$ ;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;
- print the result of  $k$ -th iteration; END FOR

Convergence check:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check:

- IF  $\text{norm}(\alpha*p_k)/\text{norm}(x_{k+1}) < \text{tol\_stag}$ 
  1. compute  $r=b-A*x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Stag\_Check ) restart;
- END IF

Residual check:

- IF  $\text{norm}(r_{k+1})/\text{norm}(b) < \text{tol}$ 
  1. compute the real residual  $r = b - A*x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Res\_Check ) restart;
- END IF

Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [spcg.c](#) for a safer version

## 10.58.2 Function Documentation

10.58.2.1 `INT fasp_solver_bdcsr_pcg ( block_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="#">block_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 precondition: the 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 665 of file pcg.c.

**10.58.2.2** `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> : 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 precondition: the 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

05/26/2014

Definition at line 373 of file pcg.c.

10.58.2.3 `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> : 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 precondition: the 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 84 of file pcg.c.

**10.58.2.4** `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> : 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 precondition: the 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 957 of file pcg.c.

## 10.59 pcg\_mf.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient (matrix free)

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

### Functions

- `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  $Au=b$ .*

### 10.59.1 Detailed Description

Krylov subspace methods – Preconditioned conjugate gradient (matrix free)

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 is:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check is like following:

- IF  $\text{norm}(\alpha p_k)/\text{norm}(x_{k+1}) < \text{tol\_stag}$ 
  1. compute  $r = b - Ax_{k+1}$ ;
  2. convergence check;



- 3. IF ( not converged & restart\_number < Max\_Stag\_Check ) restart;
- END IF

Residual check is like following:

- IF norm(r\_{k+1})/norm(b) < 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

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

### 10.59.2 Function Documentation

**10.59.2.1** 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> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 86 of file pcg\_mf.c.

## 10.60 pgcg.c File Reference

Krylov subspace methods – Preconditioned Generalized CG.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "itsolver_util.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 genirilzed conjugate gradient (GCG) method for solving  $Au=b$ .*

### 10.60.1 Detailed Description

Krylov subspace methods – Preconditioned Generalized CG.

#### Note

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

### 10.60.2 Function Documentation

**10.60.2.1 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 genirilzed conjugate gradient (GCG) method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <b>dCSRmat</b> : the coefficient matrix
<i>b</i>	Pointer to <b>dvector</b> : the right hand side
<i>u</i>	Pointer to <b>dvector</b> : the unknowns
<i>pc</i>	Pointer to <b>precond</b> : the 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 44 of file pgcg.c.

## 10.61 pgcg\_mf.c File Reference

Krylov subspace methods – Preconditioned Generalized CG (matrix free)

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

### Functions

- [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  $Au=b$ .*

#### 10.61.1 Detailed Description

Krylov subspace methods – Preconditioned Generalized CG (matrix free)

##### Note

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

#### 10.61.2 Function Documentation

10.61.2.1 [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  $Au=b$ .

##### Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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
<i>prtlvl</i>	How much information to print out

### Returns

Iteration number if converges; ERROR otherwise.

### Author

Xiaozhe Hu

### Date

01/01/2012

### Note

Not completely implemented yet! –Chensong

Modified by Chensong Zhang on 05/01/2012 Modified by Feiteng Huang on 09/26/2012: matrix free

Definition at line 47 of file pgcg\_mf.c.

## 10.62 pgcr.c File Reference

Krylov subspace methods – Preconditioned GCR.

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

### Functions

- `INT fasp_solver_dcsr_pgcr` (`dCSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, const `REAL` `tol`, const `INT` `MaxIt`, const `SHORT` `restart`, const `SHORT` `stop_type`, const `SHORT` `prtlvl`)  
*A preconditioned GCR method for solving  $Au=b$ .*
- `INT fasp_solver_dcsr_pgcr1` (`dCSRmat *A`, `dvector *b`, `dvector *x`, `precond *pc`, const `REAL` `tol`, const `INT` `MaxIt`, const `SHORT` `restart`, const `SHORT` `stop_type`, const `SHORT` `prtlvl`)  
*A preconditioned GCR method for solving  $Au=b$ .*

#### 10.62.1 Detailed Description

Krylov subspace methods – Preconditioned GCR.

## 10.62.2 Function Documentation

10.62.2.1 `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 the coefficient matrix
<i>b</i>	Pointer to the dvector of right hand side
<i>x</i>	Pointer to the dvector of dofs
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopage
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restart number for GCR
<i>stop_type</i>	Stopping type
<i>prtlvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Zheng Li

## Date

12/23/2014

Definition at line 37 of file pgcr.c.

**10.62.2.2** `INT fasp_solver_dcsr_pgcr1 ( 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 the coefficient matrix
<i>b</i>	Pointer to the dvector of right hand side
<i>x</i>	Pointer to the dvector of dofs
<i>pc</i>	Pointer to the structure of precondition (precond)
<i>tol</i>	Tolerance for stopage
<i>MaxIt</i>	Maximal number of iterations
<i>restart</i>	Restart number for GCR
<i>stop_type</i>	Stopping type
<i>prtlvl</i>	How much information to print out

## Returns

Iteration number if converges; ERROR otherwise.

## Author

Lu Wang

## Date

11/02/2014

## Warning

Deprecated function. Remove it later!!! –Chensong

Definition at line 226 of file pgcr.c.

## 10.63 pgmres.c File Reference

Krylov subspace methods – Right-preconditioned GMRes.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "itsolver_util.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\\_bdcsr\\_pgmres](#) ([block\\_dCSRmat](#) \*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  $Au=b$ .*

### 10.63.1 Detailed Description

Krylov subspace methods – Right-preconditioned GMRes.

## Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
 Four subroutines use the same algorithm for different matrix types!  
 See also [pvgmres.c](#) for a variable restarting version.  
 See [spgmres.c](#) for a safer version

### 10.63.2 Function Documentation

10.63.2.1 [INT fasp\\_solver\\_bdcsr\\_pgmres](#) ( [block\\_dCSRmat](#) \* 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  $Au=b$ .

## Parameters

<i>A</i>	Pointer to <a href="#">block_dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 356 of file pgmres.c.

10.63.2.2 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> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 659 of file pgmres.c.

**10.63.2.3** `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> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 53 of file pgmres.c.

**10.63.2.4** `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> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 963 of file pgmres.c.

## 10.64 pgmres\_mf.c File Reference

Krylov subspace methods – Preconditioned GMRes (matrix free)

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

## Functions

- [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](#) prtlvl)  
*Solve "Ax=b" using PGMRES (right preconditioned) iterative method.*

### 10.64.1 Detailed Description

Krylov subspace methods – Preconditioned GMRes (matrix free)

## Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
Refer to A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMR↔ES(m) Journal of Computational and Applied Mathematics, 230 (2009) pp. 751-761. UCRL-JRNL-235266.

## 10.64.2 Function Documentation

10.64.2.1 **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** *prtlvl* )

Solve "Ax=b" using PGMRES (right preconditioned) iterative method.

### Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 50 of file pgmres\_mf.c.

## 10.65 pminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual.

```
#include <math.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "itsolver_util.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_bdcsr_pminres` (`block_dCSRmat *A`, `dvector *b`, `dvector *u`, `precond *pc`, `const REAL tol`, `const INT MaxIt`, `const SHORT stop_type`, `const SHORT prtvl`)

*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 prtvl`)

*A preconditioned minimal residual (Minres) method for solving  $Au=b$ .*

### 10.65.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual.

Abstract algorithm of Krylov method

Krylov method to solve  $Ax=b$  is to generate  $\{x_k\}$  to approximate  $x$ , where  $x_k$  is the optimal solution in Krylov space

$V_k = \text{span}\{r_0, A*r_0, A^2*r_0, \dots, A^{k-1}*r_0\}$ ,

under some inner product.

For the implementation, we generate a series of  $\{p_k\}$  such that  $V_k = \text{span}\{p_1, \dots, p_k\}$ . Details:

Step 0. Given  $A$ ,  $b$ ,  $x_0$ ,  $M$

Step 1. Compute residual  $r_0 = b - A*x_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1}*r_0$ ,  $p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:\text{MaxIt}$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha*p_k$ ;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;
- print the result of  $k$ -th iteration; END FOR

Convergence check:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check:

- IF  $\text{norm}(\alpha*p_k)/\text{norm}(x_{k+1}) < \text{tol\_stag}$ 
  1. compute  $r = b - A*x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Stag\_Check ) restart;
- END IF

Residual check:

- IF  $\text{norm}(r_{k+1})/\text{norm}(b) < \text{tol}$ 
  1. compute the real residual  $r = b - A * x_{k+1}$ ;
  2. convergence check;
  3. IF ( not converged & restart\_number < Max\_Res\_Check ) restart;
- END IF

**Note**

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [spminres.c](#) for a safer version

**10.65.2 Function Documentation**

**10.65.2.1** INT fasp\_solver\_bdcsr\_pminres ( block\_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="#">block_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 precondition: the 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 499 of file pminres.c.

**10.65.2.2** 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> : 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 precondition: the 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 92 of file pminres.c.

**10.65.2.3** `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> : 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 precondition: the 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 902 of file pminres.c.

## 10.66 pminres\_mf.c File Reference

Krylov subspace methods – Preconditioned minimal residual (matrix free)

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

### Functions

- [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  $Au=b$ .*

#### 10.66.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual (matrix free)

Abstract algorithm of Krylov method

Krylov method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$ , where  $x_k$  is the optimal solution in Krylov space $V_k = \text{span}\{r_0, A*r_0, A^2*r_0, \dots, A^{k-1}*r_0\},$ 

under some inner product.

For the implementation, we generate a series of  $\{p_k\}$  such that  $V_k = \text{span}\{p_1, \dots, p_k\}$ . Details:Step 0. Given  $A, b, x_0, M$ Step 1. Compute residual  $r_0 = b - A*x_0$  and convergence check;Step 2. Initialization  $z_0 = M^{-1}*r_0, p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:\text{MaxIt}$ 

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha*p_k$ ;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- perform residual check;
- obtain  $p_{k+1}$  using  $\{p_0, p_1, \dots, p_k\}$ ;
- prepare for next iteration;

- print the result of k-th iteration; END FOR

Convergence check is:  $\text{norm}(r)/\text{norm}(b) < \text{tol}$

Stagnation check is like following:

- 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 is like following:

- 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

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

## 10.66.2 Function Documentation

10.66.2.1 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> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 89 of file pminres\_mf.c.

## 10.67 precondition\_bcsr.c File Reference

Preconditioners for [block\\_dCSRmat](#) 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*

### 10.67.1 Detailed Description

Preconditioners for [block\\_dCSRmat](#) matrices.

## 10.67.2 Function Documentation

10.67.2.1 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 26 of file precondition\_bcsr.c.

**10.67.2.2 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 101 of file precondition\_bcsr.c.

**10.67.2.3 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 166 of file precondition\_bcsr.c.

10.67.2.4 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 252 of file precondition\_bcsr.c.

**10.67.2.5 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 334 of file precondition\_bcsr.c.

**10.67.2.6 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 408 of file precondition\_bcsr.c.

10.67.2.7 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 669 of file precondition\_bcsr.c.

**10.67.2.8 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 788 of file precondition\_bcsr.c.

**10.67.2.9 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 506 of file precondition\_bcsr.c.

10.67.2.10 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 588 of file precondition\_bcsr.c.

10.67.2.11 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 898 of file precondition\_bcsr.c.

## 10.68 precondition\_bsr.c File Reference

Preconditioners for dBSRmat matrices.

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

## Functions

- void [fasp\\_precond\\_dbsr\\_diag](#) (REAL \**r*, REAL \**z*, void \**data*)  
*Diagonal preconditioner  $z = \text{inv}(D) * r$ .*
- void [fasp\\_precond\\_dbsr\\_diag\\_nc2](#) (REAL \**r*, REAL \**z*, void \**data*)  
*Diagonal preconditioner  $z = \text{inv}(D) * r$ .*

- 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_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.*

### 10.68.1 Detailed Description

Preconditioners for `dBSRmat` matrices.

### 10.68.2 Function Documentation

#### 10.68.2.1 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 563 of file `precond_bsr.c`.

#### 10.68.2.2 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 643 of file precondition\_bsr.c.

**10.68.2.3** 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 37 of file precondition\_bsr.c.

**10.68.2.4** 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 111 of file `precond_bsr.c`.

10.68.2.5 `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 161 of file `precond_bsr.c`.

10.68.2.6 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 211 of file `precond_bsr.c`.

10.68.2.7 `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 260 of file `precond_bsr.c`.

10.68.2.8 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 306 of file `precond_bsr.c`.

**10.68.2.9** `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 607 of file `precond_bsr.c`.

## 10.69 `precond_csr.c` File Reference

Preconditioners for [dCSRmat](#) matrices.

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



## Functions

- `precond * fasp_precond_setup` (const `SHORT` `precond_type`, `AMG_param` \*`amgparam`, `ILU_param` \*`iluparam`, `dCSRmat` \*`A`)  
*#include "forts\_ns.h"*
- 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*

### 10.69.1 Detailed Description

Preconditioners for `dCSRmat` matrices.

### 10.69.2 Function Documentation

#### 10.69.2.1 void fasp\_precond\_amg ( `REAL` \* `r`, `REAL` \* `z`, void \* `data` )

AMG preconditioner.

##### Parameters

<code>r</code>	Pointer to the vector needs preconditioning
<code>z</code>	Pointer to preconditioned vector
<code>data</code>	Pointer to precondition data

##### Author

Chensong Zhang

## Date

04/06/2010

Definition at line 400 of file precondition\_csr.c.

**10.69.2.2 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 535 of file precondition\_csr.c.

**10.69.2.3 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 469 of file precondition\_csr.c.

**10.69.2.4 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 159 of file precondition\_csr.c.

10.69.2.5 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 436 of file precondition\_csr.c.

10.69.2.6 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 619 of file precondition\_csr.c.

10.69.2.7 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 185 of file precondition\_csr.c.

10.69.2.8 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 302 of file precondition\_csr.c.

10.69.2.9 void fasp\_precond\_ilu\_forward ( REAL \* *r*, REAL \* *z*, void \* *data* )

ILU preconditioner: only forward sweep.

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu, Shiquan Zhang

## Date

04/06/2010

Definition at line 249 of file precondition\_csr.c.

10.69.2.10 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 502 of file precondition\_csr.c.

10.69.2.11 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 355 of file precondition\_csr.c.

10.69.2.12 precondition \* fasp\_precond\_setup ( const SHORT *precond\_type*, AMG\_param \* *amgparam*, ILU\_param \* *iluparam*, dCSRmat \* *A* )

#include "forts\_ns.h"

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 32 of file `precond_csr.c`.

## 10.70 `precond_str.c` File Reference

Preconditioners for `dSTRmat` matrices.

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

## Functions

- void `fasp_precond_dstr_diag` (`REAL *r`, `REAL *z`, void \*data)  
*Diagonal preconditioner  $z = \text{inv}(D) * r$ .*
- void `fasp_precond_dstr_ilu0` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition.*
- void `fasp_precond_dstr_ilu1` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition.*
- void `fasp_precond_dstr_ilu0_forward` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition:  $Lz = r$ .*
- void `fasp_precond_dstr_ilu0_backward` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(0) decomposition:  $Uz = r$ .*
- void `fasp_precond_dstr_ilu1_forward` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition:  $Lz = r$ .*
- void `fasp_precond_dstr_ilu1_backward` (`REAL *r`, `REAL *z`, void \*data)  
*Preconditioning using STR\_ILU(1) decomposition:  $Uz = r$ .*
- void `fasp_precond_dstr_blockgs` (`REAL *r`, `REAL *z`, void \*data)  
*CPR-type preconditioner (STR format)*

### 10.70.1 Detailed Description

Preconditioners for [dSTRmat](#) matrices.

### 10.70.2 Function Documentation

#### 10.70.2.1 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 1706 of file precondition\_str.c.

#### 10.70.2.2 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 27 of file precondition\_str.c.

#### 10.70.2.3 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 54 of file precondition\_str.c.

10.70.2.4 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 978 of file precondition\_str.c.

10.70.2.5 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 815 of file precondition\_str.c.



10.70.2.6 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 336 of file precondition\_str.c.

10.70.2.7 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 1425 of file precondition\_str.c.

10.70.2.8 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 1159 of file precondition\_str.c.

## 10.71 pvfgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restarting flexible GMRes.

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

### Functions

- **INT fasp\_solver\_dcsr\_pvfgmres** (**dCSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, const **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtIvl)  
Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during the iteration and flexible preconditioner can be used.
- **INT fasp\_solver\_dbsr\_pvfgmres** (**dBSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, const **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtIvl)  
Solve "Ax=b" using PFGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during the iteration and flexible preconditioner can be used.
- **INT fasp\_solver\_bdcsr\_pvfgmres** (**block\_dCSRmat** \*A, **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 PFGMRES (right preconditioned) iterative method in which the restart parameter can be adaptively modified during the iteration and flexible preconditioner can be used.

### 10.71.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restarting flexible GMRes.

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
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.  
This file is modified from [pvfgmres.c](#)

### 10.71.2 Function Documentation

**10.71.2.1 INT fasp\_solver\_bdcsr\_pvfgmres ( block\_dCSRmat \* A, dvector \* b, dvector \* x, precondition \* pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop\_type, const SHORT prtIvl )**

Solve "Ax=b" using PFGMRES (right preconditioned) iterative method in which the restart parameter can be adaptively modified during the iteration and flexible preconditioner can be used.

#### Parameters

*A	pointer to the coefficient matrix
*b	pointer to the right hand side vector

<i>*x</i>	pointer to the 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. $\ r_k\ /\ r_0\  < tol$ , 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 712 of file pvfgmres.c.

**10.71.2.2 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 the iteration and flexible preconditioner can be used.

#### 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>pc</i>	Pointer to precondition: the 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 382 of file pvfgmres.c.

**10.71.2.3** `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 the iteration and flexible preconditioner can be used.

## 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>pc</i>	Pointer to precondition: the 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 54 of file pvfgmres.c.

## 10.72 pvfgmres\_mf.c File Reference

Krylov subspace methods – Preconditioned variable-restarting flexible GMRes (matrix free)

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

## Functions

- `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 the iteration and flexible preconditioner can be used.*

### 10.72.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restarting flexible GMRes (matrix free)

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM

Refer to A.H. Baker, E.R. Jessup, and Tz.V. Kolev A Simple Strategy for Varying the Restart Parameter in GMR↔ES(m) Journal of Computational and Applied Mathematics, 230 (2009) pp. 751-761. UCRL-JRNL-235266.

This file is modified from [pvgmres.c](#)

### 10.72.2 Function Documentation

**10.72.2.1** `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 the iteration and flexible preconditioner can be used.

#### Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 55 of file pvfgmres\_mf.c.

## 10.73 pvgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRes.

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

### Functions

- **INT fasp\_solver\_dcsr\_pvgmres** (**dCSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, const **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during the iteration.*
- **INT fasp\_solver\_bdcsr\_pvgmres** (**block\_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 the iteration.*
- **INT fasp\_solver\_dbsr\_pvgmres** (**dBSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, const **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during the iteration.*
- **INT fasp\_solver\_dstr\_pvgmres** (**dSTRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, const **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Right preconditioned GMRES method in which the restart parameter can be adaptively modified during the iteration.*

### 10.73.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restart GMRes.

#### Note

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 [spvgmres.c](#) for a safer version

### 10.73.2 Function Documentation

**10.73.2.1 INT fasp\_solver\_bdcsr\_pvgmres** ( **block\_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 the iteration.

#### Parameters

<i>A</i>	Pointer to <b>dCSRmat</b> : the coefficient matrix
<i>b</i>	Pointer to <b>dvector</b> : the right hand side
<i>x</i>	Pointer to <b>dvector</b> : the unknowns

<i>pc</i>	Pointer to precondition: the 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 393 of file pvgmres.c.

**10.73.2.2** `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 the iteration.

**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>pc</i>	Pointer to precondition: the 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 738 of file pvgmres.c.



10.73.2.3 `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 the iteration.

## 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>pc</i>	Pointer to precondition: the 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 51 of file pvgmres.c.

**10.73.2.4** `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 the iteration.

## 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>pc</i>	Pointer to precondition: the 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 1083 of file pvgmres.c.

## 10.74 pvgmres\_mf.c File Reference

Krylov subspace methods – Preconditioned variable-restarting GMRes (matrix free)

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

### Functions

- [INT fasp\\_solver\\_pvgmres](#) ([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 the iteration.*

#### 10.74.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restarting GMRes (matrix free)

## Note

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.

#### 10.74.2 Function Documentation

10.74.2.1 [INT fasp\\_solver\\_pvgmres](#) ( [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 the iteration.

## Parameters

<i>mf</i>	Pointer to <a href="#">mxv_matfree</a> : the spmv operation
<i>b</i>	Pointer to dvector: the right hand side
<i>x</i>	Pointer to dvector: the unknowns
<i>pc</i>	Pointer to precondition: the 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 54 of file pvgmres\_mf.c.

## 10.75 quadrature.c File Reference

Quadrature rules.

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

**Functions**

- void [fasp\\_quad2d](#) (const [INT](#) num\_qp, const [INT](#) ncoor, [REAL](#)(\*quad)[3])  
*Initialize Lagrange quadrature points and weights.*
- void [fasp\\_gauss2d](#) (const [INT](#) num\_qp, const [INT](#) ncoor, [REAL](#)(\*gauss)[3])  
*Initialize Gauss quadrature points and weights.*

### 10.75.1 Detailed Description

Quadrature rules.

### 10.75.2 Function Documentation

10.75.2.1 void [fasp\\_gauss2d](#) ( const [INT](#) num\_qp, const [INT](#) ncoor, [REAL](#)(\*) gauss[3] )

Initialize Gauss quadrature points and weights.

## Parameters

<i>num_qp</i>	Number of quadrature points
<i>ncoor</i>	Dimension of space
<i>gauss</i>	Quadrature points and weight

## Author

Xuehai Huang, Chensong Zhang, Ludmil Zikatanov

## Date

10/21/2008

## Note

`gauss[*][0]` – quad point x in ref coor `gauss[*][1]` – quad point y in ref coor `gauss[*][2]` – quad weight

Definition at line 210 of file `quadrature.c`.

**10.75.2.2** `void fasp_quad2d ( const INT num_qp, const INT ncoor, REAL(*) quad[3] )`

Initialize Lagrange quadrature points and weights.

## Parameters

<i>num_qp</i>	Number of quadrature points
<i>ncoor</i>	Dimension of space
<i>quad</i>	Quadrature points and weights

## Author

Xuehai Huang, Chensong Zhang, Ludmil Zikatanov

## Date

10/21/2008

## Note

`quad[*][0]` – quad point x in ref coor `quad[*][1]` – quad point y in ref coor `quad[*][2]` – quad weight

Definition at line 31 of file `quadrature.c`.

## 10.76 rap.c File Reference

Tripple-matrix multiplication  $R \cdot A \cdot P$ .

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

## Functions

- `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  $R \cdot A \cdot P$ .*

### 10.76.1 Detailed Description

Tripple-matrix multiplication  $R \cdot A \cdot P$ .

C-version by Ludmil Zikatanov 2010-04-08

tested 2010-04-08

### 10.76.2 Function Documentation

10.76.2.1 `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`

Definition at line 33 of file rap.c.

## 10.77 schwarz\_setup.c File Reference

Setup phase for the Schwarz methods.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.h"
#include "mg_util.inl"
```

## Functions

- void `fasp_Schwarz_get_block_matrix (Schwarz_data *Schwarz, INT nblk, INT *iblock, INT *jblock, INT *mask)`  
*Form Schwarz partition data.*

- `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.*

### 10.77.1 Detailed Description

Setup phase for the Schwarz methods.

### 10.77.2 Function Documentation

10.77.2.1 `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 405 of file schwarz\_setup.c.

10.77.2.2 `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 295 of file schwarz\_setup.c.

10.77.2.3 void fasp\_Schwarz\_get\_block\_matrix ( Schwarz\_data \* Schwarz, INT nblk, INT \* iblock, INT \* jblock, INT \* mask )

Form Schwarz partition data.

**Parameters**

<i>Schwarz</i>	Pointer to the Schwarz data
<i>nblk</i>	Number of partitions
<i>iblock</i>	Pointer to number of vertices on each level
<i>jblock</i>	Pointer to vertices of each level
<i>mask</i>	Pointer to flag array

**Author**

Zheng Li, Chensong Zhang

**Date**

2014/09/29

Definition at line 35 of file schwarz\_setup.c.

10.77.2.4 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 126 of file schwarz\_setup.c.



## 10.78 smat.c File Reference

Simple operations for *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\\_blas\\_smat\\_inv\\_nc2](#) (REAL \*a)  
*Compute the inverse matrix of a 2\*2 full matrix A (in place)*
- void [fasp\\_blas\\_smat\\_inv\\_nc3](#) (REAL \*a)  
*Compute the inverse matrix of a 3\*3 full matrix A (in place)*
- void [fasp\\_blas\\_smat\\_inv\\_nc4](#) (REAL \*a)  
*Compute the inverse matrix of a 4\*4 full matrix A (in place)*
- void [fasp\\_blas\\_smat\\_inv\\_nc5](#) (REAL \*a)  
*Compute the inverse matrix of a 5\*5 full matrix A (in place)*
- void [fasp\\_blas\\_smat\\_inv\\_nc7](#) (REAL \*a)  
*Compute the inverse matrix of a 7\*7 matrix a.*
- void [fasp\\_blas\\_smat\\_inv\\_nc](#) (REAL \*a, const INT n)  
*Compute the inverse of a matrix using Gauss Elimination.*
- void [fasp\\_blas\\_smat\\_invp\\_nc](#) (REAL \*a, const INT n)  
*Compute the inverse of a matrix using Gauss Elimination with Pivoting.*
- INT [fasp\\_blas\\_smat\\_inv](#) (REAL \*a, const INT n)  
*Compute the inverse matrix of a small full matrix a.*
- REAL [fasp\\_blas\\_smat\\_Linfinity](#) (REAL \*A, const INT n)  
*Compute the L infinity norm of A.*
- void [fasp\\_iden\\_free](#) (idenmat \*A)  
*Free idenmat sparse matrix data memeory space.*
- 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.*

### 10.78.1 Detailed Description

Simple operations for *small* dense matrices in row-major format.

## 10.78.2 Macro Definition Documentation

10.78.2.1 `#define SWAP( a, b ) {temp=(a);(a)=(b);(b)=temp;}`

swap two numbers

Definition at line 9 of file smat.c.

## 10.78.3 Function Documentation

10.78.3.1 `INT fasp_blas_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 554 of file smat.c.

10.78.3.2 `void fasp_blas_smat_inv_nc ( REAL * a, const INT n )`

Compute the inverse of a matrix using Gauss Elimination.

Parameters

<i>a</i>	Pointer to the REAL array which stands a n*n matrix
<i>n</i>	Dimension of the matrix

Author

Xiaozhe Hu, Shiquan Zhang

Date

05/01/2010

Definition at line 405 of file smat.c.

10.78.3.3 `void fasp_blas_smat_inv_nc2 ( REAL * a )`

Compute the inverse matrix of a 2\*2 full matrix A (in place)

## Parameters

<i>a</i>	Pointer to the REAL array which stands a 2*2 matrix
----------	---

## Author

Xiaozhe Hu

## Date

18/11/2011

Definition at line 25 of file smat.c.

**10.78.3.4 void fasp\_blas\_smat\_inv\_nc3 ( REAL \* *a* )**

Compute the inverse matrix of a 3\*3 full matrix A (in place)

## Parameters

<i>a</i>	Pointer to the REAL array which stands a 3*3 matrix
----------	---

## Author

Xiaozhe Hu, Shiquan Zhang

## Date

05/01/2010

Definition at line 61 of file smat.c.

**10.78.3.5 void fasp\_blas\_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 115 of file smat.c.

**10.78.3.6 void fasp\_blas\_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 173 of file smat.c.

10.78.3.7 void fasp\_blas\_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 389 of file smat.c.

10.78.3.8 void fasp\_blas\_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 472 of file smat.c.

**10.78.3.9 REAL fasp\_blas\_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 595 of file smat.c.

**10.78.3.10 void fasp\_iden\_free ( idenmat \* A )**

Free idenmat sparse matrix data memeory space.

**Parameters**

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

**Author**

Chensong Zhang

**Date**

2010/04/03

Definition at line 628 of file smat.c.

**10.78.3.11 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 728 of file smat.c.

10.78.3.12 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 648 of file smat.c.

**10.78.3.13 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 665 of file smat.c.

**10.78.3.14 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 682 of file smat.c.

**10.78.3.15 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 703 of file smat.c.

## 10.79 smoother\_bsr.c File Reference

Smoothers for [dBSRmat](#) matrices.

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

### Functions

- void [fasp\\_smoother\\_dbsr\\_jacobi](#) ([dBSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*u)
 

*Jacobi relaxation.*
- void [fasp\\_smoother\\_dbsr\\_jacobi\\_setup](#) ([dBSRmat](#) \*A, [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.*

### 10.79.1 Detailed Description

Smoothers for `dBSRmat` matrices.

### 10.79.2 Function Documentation

10.79.2.1 void `fasp_smoother_dbsr_gs` ( `dBSRmat * A`, `dvector * b`, `dvector * u`, `INT order`, `INT * mark` )

Gauss-Seidel relaxation.

Parameters

<i>A</i>	Pointer to <code>dBSRmat</code> : the coefficient matrix
<i>b</i>	Pointer to <code>dvector</code> : the right hand side
<i>u</i>	Pointer to <code>dvector</code> : the unknowns (IN: initial, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If <code>mark = NULL</code> ASCEND 12: in ascending order DE↔SCEND 21: in descending order If <code>mark != NULL</code> : 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 411 of file `smoother_bsr.c`.

10.79.2.2 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 DE↔ SCEND 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 531 of file smoother\_bsr.c.

10.79.2.3 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 568 of file smoother\_bsr.c.

10.79.2.4 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 641 of file smoother\_bsr.c.

**10.79.2.5** void fasp\_smoother\_dbsr\_gs\_descend ( dBSRmat \* *A*, dvector \* *b*, dvector \* *u*, REAL \* *diagin*v )

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>diagin</i> v	Inverses for all the diagonal blocks of A

**Author**

Zhiyang Zhou

**Date**

2010/10/25

Definition at line 712 of file smoother\_bsr.c.

**10.79.2.6** void fasp\_smoother\_dbsr\_gs\_descend1 ( dBSRmat \* *A*, dvector \* *b*, dvector \* *u* )

Gauss-Seidel relaxation in the descending order.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)

**Author**

Xiaozhe Hu

## Date

01/01/2014

## Note

The only difference between the functions 'fasp\_smoother\_dbsr\_gs\_ascend1' and 'fasp\_smoother\_dbsr\_gs\_↔ascend' is that we don't have to multiply by the inverses of the diagonal blocks in each ROW since matrix A has been such scaled that all the diagonal blocks become identity matrices.

Definition at line 786 of file smoother\_bsr.c.

10.79.2.7 void fasp\_smoother\_dbsr\_gs\_order1 ( dBSRmat \* A, dvector \* b, dvector \* u, REAL \* diaginv, INT \* mark )

Gauss-Seidel relaxation in the user-defined order.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A
<i>mark</i>	Pointer to the user-defined ordering

## Author

Zhiyang Zhou

## Date

2010/10/25

Definition at line 858 of file smoother\_bsr.c.

10.79.2.8 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 936 of file smoother\_bsr.c.

**10.79.2.9** 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

**Date**

2010/10/25 Adjust the work space of ilu smoother by Zheng Li 04/26/2015.

form residual  $zr = b - A x$

solve LU  $z=zr$

$x=x+z$

Definition at line 1566 of file smoother\_bsr.c.

**10.79.2.10** 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 33 of file smoother\_bsr.c.

10.79.2.11 void fasp\_smoother\_dbsr\_jacobi1 ( dBSRmat \* *A*, dvector \* *b*, dvector \* *u*, REAL \* *diaginv* )

Jacobi relaxation.

## Parameters

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>diaginv</i>	Inverses for all the diagonal blocks of A

## Author

Zhiyang Zhou

## Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/03/2012

Definition at line 257 of file smoother\_bsr.c.

10.79.2.12 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.

## 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 148 of file smoother\_bsr.c.

10.79.2.13 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 DE↔ SCEND 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 1013 of file smoother\_bsr.c.

10.79.2.14 void fasp\_smoother\_dbsr\_sor1 ( dBSRmat \* *A*, dvector \* *b*, dvector \* *u*, INT *order*, INT \* *mark*, REAL \* *diaginv*, REAL *weight* )

SOR relaxation.

**Parameters**

<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns (IN: initial guess, OUT: approximation)
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending order DE↔ SCEND 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 1135 of file smoother\_bsr.c.

10.79.2.15 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 1176 of file smoother\_bsr.c.

```
10.79.2.16 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 1305 of file smoother\_bsr.c.

```
10.79.2.17 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.

## 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>diagin</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 1436 of file smoother\_bsr.c.

## 10.80 smoother\_csr.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.*

### 10.80.1 Detailed Description

Smoothers for `dCSRmat` matrices.

### 10.80.2 Function Documentation

10.80.2.1 void `fasp_smoother_dcsr_gs` ( `dvector` \* u, const `INT` i\_1, const `INT` i\_n, const `INT` s, `dCSRmat` \* A, `dvector` \* b, `INT` L )

Gauss-Seidel method as a smoother.

Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i_1</i>	Starting index
<i>i_n</i>	Ending index
<i>s</i>	Increasing step
<i>A</i>	Pointer to <code>dBSRmat</code> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations

Author

Xuehai Huang, Chensong Zhang

Date

09/26/2009

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

Definition at line 195 of file smoother\_csr.c.

10.80.2.2 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 364 of file smoother\_csr.c.

10.80.2.3 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 1425 of file smoother\_csr.c.

10.80.2.4 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 1067 of file smoother\_csr.c.

**10.80.2.5** 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.

## Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>i_1</i>	Starting index
<i>i_n</i>	Ending index
<i>s</i>	Increasing step
<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations

## Author

Xuehai Huang, Chensong Zhang

## Date

09/26/2009

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

Definition at line 59 of file smoother\_csr.c.

**10.80.2.6** 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)
$i_1$	Starting index
$i_n$	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

2010/11/12

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

Definition at line 1145 of file smoother\_csr.c.

10.80.2.7 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)
$i_1$	Starting index
$i_n$	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 1286 of file smoother\_csr.c.

10.80.2.8 void fasp\_smoother\_dcsr\_sgs ( dvector \*  $u$ , dCSRmat \*  $A$ , dvector \*  $b$ , INT  $L$  )

Symmetric Gauss-Seidel method as a smoother.

## Parameters

$u$	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$A$	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
$b$	Pointer to dvector: the right hand side
$L$	Number of iterations

## Author

Xiaozhe Hu

## Date

10/26/2010

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

Definition at line 629 of file smoother\_csr.c.

10.80.2.9 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.

## Parameters

$u$	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
$i\_1$	Starting index
$i\_n$	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 745 of file smoother\_csr.c.

10.80.2.10 void fasp\_smoother\_dcsr\_sor\_cf ( dvector \*  $u$ , dCSRmat \*  $A$ , dvector \*  $b$ , INT  $L$ , const REAL  $w$ , INT \*  $mark$ , const INT  $order$  )

SOR smoother with C/F ordering for  $Au=b$ .

## Parameters

<i>u</i>	Pointer to dvector: the unknowns (IN: initial, OUT: approximation)
<i>A</i>	Pointer to <a href="#">dBSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>L</i>	Number of iterations
<i>w</i>	Over-relaxation weight
<i>mark</i>	C/F marker array
<i>order</i>	C/F ordering: -1: F-first; 1: C-first

## Author

Zhiyang Zhou

## Date

2010/11/12

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

Definition at line 873 of file smoother\_csr.c.

## 10.81 smoother\_csr\_cr.c File Reference

Smoothers for [dCSRmat](#) matrices using compatible relaxation.

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

### Functions

- void [fasp\\_smoother\\_dcsr\\_gscr](#) (INT pt, INT n, REAL \*u, INT \*ia, INT \*ja, REAL \*a, REAL \*b, INT L, INT \*CF)  
*Gauss Seidel method restriced to a block.*

#### 10.81.1 Detailed Description

Smoothers for [dCSRmat](#) matrices using compatible relaxation.

#### Note

Restricted-smoothers for compatible relaxation, C/F smoothing, etc.

#### 10.81.2 Function Documentation

10.81.2.1 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 restriced 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 – remove later also as MG relaxation on error eqn
<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 38 of file smoother\_csr\_cr.c.

## 10.82 smoother\_csr\_poly.c File Reference

Smoothers for [dCSRmat](#) matrices using poly. approx. to  $A^{-1}$ .

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

## Functions

- void [fasp\\_smoother\\_dcsr\\_poly](#) ([dCSRmat](#) \*Amat, [dvector](#) \*brhs, [dvector](#) \*usol, [INT](#) n, [INT](#) ndeg, [INT](#) L)  
*poly approx to  $A^{-1}$  as MG smoother*
- void [fasp\\_smoother\\_dcsr\\_poly\\_old](#) ([dCSRmat](#) \*Amat, [dvector](#) \*brhs, [dvector](#) \*usol, [INT](#) n, [INT](#) ndeg, [INT](#) L)  
*poly approx to  $A^{-1}$  as MG smoother: JK&LTZ2010*

### 10.82.1 Detailed Description

Smoothers for [dCSRmat](#) matrices using poly. approx. to  $A^{-1}$ .

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.

## 10.82.2 Function Documentation

10.82.2.1 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 48 of file smoother\_csr\_poly.c.

10.82.2.2 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 148 of file smoother\_csr\_poly.c.

## 10.83 smoother\_str.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.*

### 10.83.1 Detailed Description

Smoothers for `dSTRmat` matrices.

### 10.83.2 Function Documentation

10.83.2.1 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 1521 of file smoother\_str.c.

10.83.2.2 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 D↔ ESCEND 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 203 of file smoother\_str.c.

10.83.2.3 void fasp\_smoother\_dstr\_gs1 ( dSTRmat \* *A*, dvector \* *b*, dvector \* *u*, const INT *order*, INT \* *mark*, REAL \* *diaginv* )

Gauss-Seidel method as the smoother with diag\_inv given.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>order</i>	Flag to indicate the order for smoothing If mark = NULL ASCEND 12: in ascending manner D↔ ESCEND 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 263 of file smoother\_str.c.

**10.83.2.4** void fasp\_smoother\_dstr\_gs\_ascend ( dSTRmat \* *A*, dvector \* *b*, dvector \* *u*, REAL \* *diaginv* )

Gauss-Seidel method as the smoother in the ascending manner.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

## Author

Shiquan Zhang, Zhiyang Zhou

## Date

10/10/2010

Definition at line 308 of file smoother\_str.c.

**10.83.2.5** 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 663 of file smoother\_str.c.

10.83.2.6 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 423 of file smoother\_str.c.

10.83.2.7 void fasp\_smoother\_dstr\_gs\_order ( dSTRmat \* *A*, dvector \* *b*, dvector \* *u*, REAL \* *diaginv*, INT \* *mark* )

Gauss method as the smoother in the user-defined order.

## Parameters

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1
<i>mark</i>	Pointer to the user-defined order array

**Author**

Shiquan Zhang, Zhiyang Zhou

**Date**

10/10/2010

Definition at line 540 of file smoother\_str.c.

**10.83.2.8** 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 31 of file smoother\_str.c.

**10.83.2.9** void fasp\_smoother\_dstr\_jacobi1 ( dSTRmat \* *A*, dvector \* *b*, dvector \* *u*, REAL \* *diaginv* )

Jacobi method as the smoother with diag\_inv given.

**Parameters**

<i>A</i>	Pointer to <a href="#">dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>u</i>	Pointer to dvector: the unknowns
<i>diaginv</i>	All the inverse matrices for all the diagonal block of A when (A->nc)>1, and NULL when (A->nc)=1

**Author**

Shiquan Zhang, Zhiyang Zhou

Date

10/10/2010

Definition at line 79 of file smoother\_str.c.

10.83.2.10 void fasp\_smoother\_dstr\_schwarz ( dSTRmat \* *A*, dvector \* *b*, dvector \* *u*, dvector \* *diagin*, 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>diagin</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 1643 of file smoother\_str.c.

10.83.2.11 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 D↔ ESCEND 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 855 of file smoother\_str.c.



10.83.2.12 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 D↔ESCEND 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 916 of file smoother\_str.c.

10.83.2.13 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 962 of file smoother\_str.c.

10.83.2.14 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 1334 of file smoother\_str.c.

10.83.2.15 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 1082 of file smoother\_str.c.

10.83.2.16 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 1203 of file smoother\_str.c.

## 10.84 sparse\_block.c File Reference

Sparse matrix block operations.

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

## Functions

- void [fasp\\_bdcsr\\_free](#) ([block\\_dCSRmat](#) \*A)  
*Free block CSR sparse matrix data memory space.*
- [SHORT fasp\\_dcsr\\_getblk](#) ([dCSRmat](#) \*A, [INT](#) \*Is, [INT](#) \*Js, const [INT](#) m, const [INT](#) n, [dCSRmat](#) \*B)  
*Get a sub CSR matrix of A with specified rows and columns.*
- [SHORT fasp\\_dbsr\\_getblk](#) ([dBSRmat](#) \*A, [INT](#) \*Is, [INT](#) \*Js, const [INT](#) m, const [INT](#) n, [dBSRmat](#) \*B)  
*Get a sub BSR matrix of A with specified rows and columns.*
- [dCSRmat fasp\\_dbsr\\_getblk\\_dcsr](#) ([dBSRmat](#) \*A)  
*get dCSRmat block from a dBSRmat matrix*
- [dCSRmat fasp\\_dbsr\\_Linfinity\\_dcsr](#) ([dBSRmat](#) \*A)  
*get dCSRmat from a dBSRmat matrix using L\_infinity norm of each small block*

### 10.84.1 Detailed Description

Sparse matrix block operations.

## 10.84.2 Function Documentation

10.84.2.1 void fasp\_bdcsr\_free ( block\_dCSRmat \* A )

Free block CSR sparse matrix data memory space.

## Parameters

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

## Author

Xiaozhe Hu

## Date

04/18/2014

Definition at line 30 of file sparse\_block.c.

10.84.2.2 **SHORT** fasp\_dbsr\_getblk ( [dBSRmat](#) \* *A*, INT \* *Is*, 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 160 of file sparse\_block.c.

10.84.2.3 **dCSRmat** fasp\_dbsr\_getblk\_dcsr ( [dBSRmat](#) \* *A* )get [dCSRmat](#) block from a [dBSRmat](#) matrix

## Parameters

<i>*A</i>	Pointer to the BSR format matrix
-----------	----------------------------------

**Returns**

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

**Author**

Xiaozhe Hu

**Date**

03/16/2012

Definition at line 256 of file sparse\_block.c.

#### 10.84.2.4 **dCSRmat fasp\_dbsr\_Linfinity\_dcsr ( dBSRmat \* A )**

get [dCSRmat](#) from a [dBSRmat](#) matrix using L\_infinity norm of each small block

**Parameters**

<i>*A</i>	Pointer to the BSR format matrix
-----------	----------------------------------

**Returns**

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

**Author**

Xiaozhe Hu

**Date**

05/25/2014

Definition at line 312 of file sparse\_block.c.

#### 10.84.2.5 **SHORT fasp\_dcsr\_getblk ( dCSRmat \* A, INT \* Is, 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

$m$	Number of selected rows
$n$	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 66 of file sparse\_block.c.

**10.85 sparse\_bsr.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` (`dBSRmat` \*A, `dBSRmat` \*B)
 

copy a `dCSRmat` to a new one  $B=A$
- `INT fasp_dbsr_trans` (`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_getdiaginv` (`dBSRmat` \*A)
 

Get  $D^{-1}$  of matrix A.
- `dBSRmat fasp_dbsr_diaginv` (`dBSRmat` \*A)

- *Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.*  
**dBSRmat fasp\_dbsr\_diaginv2** (dBSRmat \*A, REAL \*diaginv)
- *Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.*  
**dBSRmat fasp\_dbsr\_diaginv3** (dBSRmat \*A, REAL \*diaginv)
- *Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.*  
**dBSRmat fasp\_dbsr\_diaginv4** (dBSRmat \*A, REAL \*diaginv)
- *Compute  $B := D^{-1} * A$ , where 'D' is the block diagonal part of A.*  
**void fasp\_dbsr\_getdiag** (INT n, dBSRmat \*A, REAL \*diag)
- *Abstract the diagonal blocks of a BSR matrix.*  
**dBSRmat fasp\_dbsr\_diagLU** (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  $DU = \text{diag}(U^{-1})$ .*

### 10.85.1 Detailed Description

Sparse matrix operations for **dBSRmat** matrices.

### 10.85.2 Function Documentation

10.85.2.1 **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

ROW	Number of rows of block
COL	Number of columns of block
NNZ	Number of nonzero blocks
nb	Dimension of each block
storage_manner	Storage manner for each sub-block
A	Pointer to new <b>dBSRmat</b> matrix

Author

Xiaozhe Hu

Date

10/26/2010

Definition at line 87 of file sparse\_bsr.c.

10.85.2.2 **void fasp\_dbsr\_cp** ( dBSRmat \* A, dBSRmat \* B )

copy a **dCSRmat** to a new one B=A



## Parameters

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

## Author

Xiaozhe Hu

## Date

08/07/2011

Definition at line 181 of file sparse\_bsr.c.

### 10.85.2.3 [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 36 of file sparse\_bsr.c.

### 10.85.2.4 [dBSRmat](#) fasp\_dbsr\_diagin ( [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 496 of file sparse\_bsr.c.

**10.85.2.5 dBSRmat fasp\_dbsr\_diaginv2 ( 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 660 of file sparse\_bsr.c.

**10.85.2.6 dBSRmat fasp\_dbsr\_diaginv3 ( 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 762 of file sparse\_bsr.c.

**10.85.2.7 dBSRmat fasp\_dbsr\_diaginv4 ( 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 1120 of file sparse\_bsr.c.

**10.85.2.8 dBSRmat fasp\_dbsr\_diagLU ( dBSRmat \* A, REAL \* DL, REAL \* DU )**Compute  $B := DL * A * DU$ . We decompose each diagonal block of A into LDU form and  $DL = \text{diag}(L^{-1})$  and  $DU = \text{diag}(U^{-1})$ .

## Parameters

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

<i>DL</i>	Pointer to the $\text{diag}(L^{-1})$
<i>DU</i>	Pointer to the $\text{diag}(U^{-1})$

**Returns**

BSR matrix after scaling

**Author**

Xiaozhe Hu

**Date**

04/02/2014

Definition at line 1449 of file sparse\_bsr.c.

**10.85.2.9** `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 1677 of file sparse\_bsr.c.

**10.85.2.10** `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 292 of file sparse\_bsr.c.

**10.85.2.11 void fasp\_dbsr\_free ( dBSRmat \* *A* )**

Free memory space for BSR format sparse matrix.

**Parameters**

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

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 133 of file sparse\_bsr.c.

**10.85.2.12 fasp\_dbsr\_getdiag ( INT *n*, 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 1411 of file sparse\_bsr.c.

**10.85.2.13 dvector fasp\_dbsr\_getdiaginv ( 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 392 of file sparse\_bsr.c.

**10.85.2.14 void fasp\_dbsr\_null ( dBSRmat \* A )**

Initialize sparse matrix on structured grid.

**Parameters**

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

**Author**

Xiaozhe Hu

**Date**

10/26/2010

Definition at line 158 of file sparse\_bsr.c.

10.85.2.15 INT fasp\_dbsr\_trans ( dBSRmat \* A, dBSRmat \* AT )

Find  $A^T$  from given dBSRmat matrix A.

Parameters

A	Pointer to the dBSRmat matrix
AT	Pointer to the transpose of dBSRmat matrix A

Author

Chunsheng FENG

Date

2011/06/08

Modified by Xiaozhe Hu (08/06/2011)

Definition at line 208 of file sparse\_bsr.c.

## 10.86 sparse\_coo.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.*

### 10.86.1 Detailed Description

Sparse matrix operations for dCOOmat matrices.

### 10.86.2 Function Documentation

10.86.2.1 void fasp\_dcoo\_alloc ( const INT m, const INT n, const INT nnz, dCOOmat \* A )

Allocate COO sparse matrix memory space.

## Parameters

$m$	Number of rows
$n$	Number of columns
$nnz$	Number of nonzeros
$A$	Pointer to the <a href="#">dCSRmat</a> matrix

## Author

Xiaozhe Hu

## Date

03/25/2013

Definition at line 62 of file sparse\_coo.c.

10.86.2.2 [dCOOmat](#) fasp\_dcoo\_create ( const INT  $m$ , const INT  $n$ , const INT  $nnz$  )

Create IJ sparse matrix data memory space.

## Parameters

$m$	Number of rows
$n$	Number of columns
$nnz$	Number of nonzeros

## Returns

A The new [dCOOmat](#) matrix

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 34 of file sparse\_coo.c.

10.86.2.3 void fasp\_dcoo\_free ( [dCOOmat](#) \*  $A$  )

Free IJ sparse matrix data memory space.

## Parameters

$A$	Pointer to the <a href="#">dCOOmat</a> matrix
-----	---

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 94 of file sparse\_coo.c.



#### 10.86.2.4 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 116 of file sparse\_coo.c.

## 10.87 sparse\_csr.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.*
- [dCSRmat fasp\\_dcsr\\_perm](#) (dCSRmat \*A, INT \*P)  
*Apply permutation of A, i.e. Aperm=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.
- void `fasp_dcsr_getdiag` (INT  $n$ , dCSRmat  $*A$ , dvector  $*diag$ )  
Get first  $n$  diagonal entries of a CSR matrix  $A$ .
- void `fasp_dcsr_getcol` (const INT  $n$ , 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  $*A$ , REAL value)  
Regularize diagonal entries of a CSR sparse matrix.
- void `fasp_icsr_cp` (iCSRmat  $*A$ , iCSRmat  $*B$ )  
Copy a iCSRmat to a new one  $B=A$ .
- void `fasp_dcsr_cp` (dCSRmat  $*A$ , dCSRmat  $*B$ )  
copy a dCSRmat to a new one  $B=A$
- void `fasp_icsr_trans` (iCSRmat  $*A$ , iCSRmat  $*AT$ )  
Find transpose of iCSRmat matrix  $A$ .
- INT `fasp_dcsr_trans` (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  $*tniz$ )
- void `fasp_dcsr_compress` (dCSRmat  $*A$ , dCSRmat  $*B$ , 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  $*A$ , REAL  $dtol$ )  
Compress a CSR matrix  $A$  IN PLACE by dropping small entries  $abs(a_{ij}) \leq dtol$ .
- void `fasp_dcsr_shift` (dCSRmat  $*A$ , INT  $offset$ )  
Re-index a REAL matrix in CSR format to make the index starting from 0 or 1.
- void `fasp_dcsr_symdiagscale` (dCSRmat  $*A$ , dvector  $*diag$ )  
Symmetric diagonal scaling  $D^{-1/2}AD^{-1/2}$ .
- dCSRmat `fasp_dcsr_sympat` (dCSRmat  $*A$ )  
Get symmetric part of a dCSRmat matrix.
- void `fasp_dcsr_multicoloring` (dCSRmat  $*A$ , INT  $*flags$ , INT  $*groups$ )  
Use the greedy multi-coloring to get color groups of the adjacency graph of  $A$ .
- void `fasp_dcsr_transz` (dCSRmat  $*A$ , INT  $*p$ , dCSRmat  $*AT$ )  
Generalized transpose of  $A$ :  $(n \times m)$  matrix given in dCSRmat format.
- dCSRmat `fasp_dcsr_permz` (dCSRmat  $*A$ , INT  $*p$ )  
Permute rows and cols of  $A$ , i.e.  $A=PAP'$  by the ordering in  $p$ .
- void `fasp_dcsr_sortz` (dCSRmat  $*A$ , const SHORT isym)  
Sort each row of  $A$  in ascending order w.r.t. column indices.

### 10.87.1 Detailed Description

Sparse matrix operations for dCSRmat matrices.

### 10.87.2 Function Documentation

#### 10.87.2.1 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 125 of file sparse\_csr.c.

10.87.2.2 void fasp\_dcsr\_compress ( dCSRmat \* *A*, dCSRmat \* *B*, 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 957 of file sparse\_csr.c.

10.87.2.3 SHORT fasp\_dcsr\_compress\_inplace ( dCSRmat \* *A*, 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 1037 of file sparse\_csr.c.

10.87.2.4 void fasp\_dcsr\_cp ( dCSRmat \* *A*, dCSRmat \* *B* )

copy a dCSRmat to a new one B=A

## Parameters

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

## Author

Chensong Zhang

## Date

04/06/2010

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

Definition at line 723 of file sparse\_csr.c.

10.87.2.5 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 34 of file sparse\_csr.c.

10.87.2.6 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 553 of file sparse\_csr.c.

**10.87.2.7** `void fasp_dcsr_free ( dCSRmat * A )`

Free CSR sparse matrix data memory space.

## Parameters

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

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 166 of file sparse\_csr.c.

**10.87.2.8** `void fasp_dcsr_getcol ( const INT n, 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 474 of file sparse\_csr.c.

10.87.2.9 void fasp\_dcsr\_getdiag ( INT *n*, 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 <i>n</i> =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 410 of file sparse\_csr.c.

10.87.2.10 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 1265 of file sparse\_csr.c.

**10.87.2.11 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 204 of file sparse\_csr.c.

**10.87.2.12 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 247 of file sparse\_csr.c.

**10.87.2.13 dCSRmat fasp\_dcsr\_permz ( dCSRmat \* A, INT \* p )**

Permute rows and cols of A, i.e. A=PAP' by the ordering in p.



## Parameters

$A$	Pointer to the original <a href="#">dCSRmat</a> matrix
$p$	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 1486 of file `sparse_csr.c`.

10.87.2.14 **SHORT** `fasp_dcsr_regdiag ( dCSRmat * A, REAL value )`

Regularize diagonal entries of a CSR sparse matrix.

## Parameters

$A$	Pointer to the <a href="#">dCSRmat</a> matrix
$value$	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 659 of file `sparse_csr.c`.

10.87.2.15 **void** `fasp_dcsr_shift ( dCSRmat * A, 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 1085 of file sparse\_csr.c.

10.87.2.16 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 358 of file sparse\_csr.c.

10.87.2.17 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 1518 of file sparse\_csr.c.

10.87.2.18 void fasp\_dcsr\_symdiagscale ( dCSRmat \* A, 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 1146 of file sparse\_csr.c.

10.87.2.19 dCSRmat fasp\_dcsr\_sympat ( dCSRmat \* A )

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

Parameters

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

Returns

symmetrized the [dCSRmat](#) matrix

Author

Xiaozhe Hu

Date

03/21/2011

Definition at line 1232 of file sparse\_csr.c.

10.87.2.20 void fasp\_dcsr\_trans ( dCSRmat \* A, dCSRmat \* AT )

Find transpose of [dCSRmat](#) matrix A.

Parameters

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

<i>AT</i>	Pointer to the transpose of <a href="#">dCSRmat</a> matrix A (output)
-----------	---

**Author**

Chensong Zhang

**Date**

04/06/2010

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

Definition at line 826 of file sparse\_csr.c.

10.87.2.21 void fasp\_dcsr\_transz ( dCSRmat \* A, INT \* p, dCSRmat \* AT )

Generalized transpose of A: (n x m) matrix given in [dCSRmat](#) format.**Parameters**

<i>A</i>	Pointer to matrix in <a href="#">dCSRmat</a> for transpose, INPUT
<i>p</i>	Permutation, INPUT
<i>AT</i>	Pointer to matrix AT = transpose(A) if p = NULL, OR AT = transpose(A)p if p is not NULL

**Note**

The storage for all pointers in AT should already be allocated, i.e. AT->IA, AT->JA and AT->val should be allocated before calling this function. If A.val=NULL, then AT->val[] is not changed.

performs AT=transpose(A)p, where p is a permutation. If p=NULL then p=I is assumed. Applying twice this procedure one gets At=transpose(transpose(A)p)p = transpose(p)Ap, which is the same A with rows and columns permuted according to p.

If A=NULL, then only transposes/permutates the structure of A.

For p=NULL, applying this two times A->AT->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 1366 of file sparse\_csr.c.

10.87.2.22 void fasp\_icsr\_cp ( 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 698 of file sparse\_csr.c.

**10.87.2.23 iCSRmat fasp\_icshr\_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 80 of file sparse\_csr.c.

**10.87.2.24 void fasp\_icshr\_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 185 of file sparse\_csr.c.

10.87.2.25 void fasp\_icsr\_null ( iCSRmat \* A )

Initialize CSR sparse matrix.

## Parameters

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

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 221 of file sparse\_csr.c.

10.87.2.26 void fasp\_icsr\_trans ( [iCSRmat](#) \* *A*, [iCSRmat](#) \* *AT* )

Find transpose of [iCSRmat](#) matrix *A*.

## Parameters

<i>A</i>	Pointer to the <a href="#">iCSRmat</a> matrix <i>A</i>
<i>AT</i>	Pointer to the <a href="#">iCSRmat</a> 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 750 of file sparse\_csr.c.

## 10.88 sparse\_csrl.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.*

### 10.88.1 Detailed Description

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

#### Note

For details of CSRL format, refer to Optimizing sparse matrix vector product computations using unroll and jam by John Mellor-Crummey and John Garvin, Tech Report Rice Univ, Aug 2002.

### 10.88.2 Function Documentation

#### 10.88.2.1 [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 30 of file sparse\_csrl.c.

#### 10.88.2.2 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 58 of file sparse\_csrl.c.

## 10.89 [sparse\\_str.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](#) ([dSTRmat](#) \*A, [dSTRmat](#) \*A1)  
*Copy a [dSTRmat](#) to a new one A1=A.*

### 10.89.1 Detailed Description

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

### 10.89.2 Function Documentation

10.89.2.1 void [fasp\\_dstr\\_alloc](#) ( const [INT](#) nx, const [INT](#) ny, const [INT](#) nz, const [INT](#) nxy, const [INT](#) ngrid, const [INT](#) nband, const [INT](#) nc, [INT](#) \* offsets, [dSTRmat](#) \* A )

Allocate STR sparse matrix memory space.

#### Parameters

<i>nx</i>	Number of grids in x direction
<i>ny</i>	Number of grids in y direction
<i>nz</i>	Number of grids in z direction
<i>nxy</i>	Number of grids in x-y plane
<i>ngrid</i>	Number of grids
<i>nband</i>	Number of off-diagonal bands
<i>nc</i>	Number of components
<i>offsets</i>	Shift from diagonal
<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix

#### Author

Shiquan Zhang, Xiaozhe Hu

#### Date

05/17/2010

Definition at line 109 of file sparse\_str.c.

10.89.2.2 void fasp\_dstr\_cp ( dSTRmat \* A, dSTRmat \* A1 )

Copy a [dSTRmat](#) to a new one A1=A.

## Parameters

<i>A</i>	Pointer to the <a href="#">dSTRmat</a> matrix
<i>A1</i>	Pointer to the <a href="#">dSTRmat</a> matrix

## Author

Zhiyang Zhou

## Date

04/21/2010

Definition at line 181 of file sparse\_str.c.

**10.89.2.3** [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 57 of file sparse\_str.c.

**10.89.2.4** 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 152 of file sparse\_str.c.

**10.89.2.5 void fasp\_dstr\_null ( dSTRmat \* *A* )**

Initialize sparse matrix on structured grid.

**Parameters**

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

**Author**

Shiquan Zhang, Xiaozhe Hu

**Date**

05/17/2010

Definition at line 25 of file sparse\_str.c.

**10.90 sparse\_util.c File Reference**

Routines for sparse matrix operations.

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_funcs.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 nonzeros.*
- 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 nonzeros.*
- 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 \*nxp, 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*

### 10.90.1 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

**Todo** Remove unwanted functions from this file. –Chensong

### 10.90.2 Function Documentation

10.90.2.1 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 the 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 272 of file sparse\_util.c.

10.90.2.2 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 124 of file sparse\_util.c.

10.90.2.3 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 53 of file sparse\_util.c.

**10.90.2.4** 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 359 of file sparse\_util.c.

**10.90.2.5** 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 431 of file sparse\_util.c.

**10.90.2.6** 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 197 of file `sparse_util.c`.

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

get the maximal independet set of a CSR matrix

## Parameters

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

## Note

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

information of A

work space

return

Definition at line 909 of file `sparse_util.c`.

#### 10.90.2.8 `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 788 of file sparse\_util.c.

**10.90.2.9** void fasp\_sparse\_rapms\_ ( INT \* *ir*, INT \* *jr*, INT \* *ia*, INT \* *ja*, INT \* *ip*, INT \* *jp*, INT \* *nin*, INT \* *ncin*, INT \* *iac*, INT \* *jac*, INT \* *maxrout* )

Calculates the nonzero structure of  $R \cdot A \cdot P$ , if jac is not null. If jac is null only finds num of nonzeros.

**Note**

:I: is input :O: is output :IO: is both

**Parameters**

<i>ir</i>	:I: array of row pointers for R
<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 514 of file sparse\_util.c.

10.90.2.10 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 648 of file sparse\_util.c.

10.90.2.11 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 595 of file sparse\_util.c.

10.90.2.12 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>	:l: indices such that $y[jy]$ is nonzero
<i>y</i>	:l: is a sparse vector.
<i>nyp</i>	:l: number of nonzeros in <i>y</i>
<i>jx</i>	:l: indices such that $x[jx]$ is nonzero
<i>x</i>	:l: is a sparse vector.
<i>nxp</i>	:l: number of nonzeros in <i>x</i>
<i>icp</i>	???
<i>s</i>	:O: $s = y^t x$ .

Definition at line 733 of file sparse\_util.c.

10.90.2.13 void fasp\_sparse\_ytxbig\_ ( INT \* *jy*, REAL \* *y*, INT \* *nyp*, REAL \* *x*, REAL \* *s* )

Calculates  $s = y^t x$ . *y*-sparse, *x* - no.

## Note

:l: is input :O: is output :lO: 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 <i>y</i>
<i>x</i>	:l: also a vector assumed to have entry for any $j=jy[i]-1$ ; for $i=1:nyp$ . This means that <i>x</i> here does not have to be sparse.
<i>s</i>	:O: $s = y^t x$ .

Definition at line 699 of file sparse\_util.c.

## 10.91 spbcgs.c File Reference

Krylov subspace methods – Preconditioned BiCGstab with safety net.

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

## Functions

- [INT fasp\\_solver\\_dcsr\\_spbcgs](#) (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  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_dbsr\\_spbcgs](#) (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  $Au=b$  with safety net.*
- [INT fasp\\_solver\\_bdcsr\\_spbcgs](#) (block\_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  $Au=b$  with safety net.*

- `INT fasp_solver_dstr_spgcs (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  $Au=b$  with safety net.*

### 10.91.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab with safety net.

Abstract algorithm

PBICGStab method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$

Note: We generate a series of  $\{p_k\}$  such that  $V_k = \text{span}\{p_1, \dots, p_k\}$ .

Step 0. Given  $A, b, x_0, M$

Step 1. Compute residual  $r_0 = b - A*x_0$  and convergence check;

Step 2. Initialization  $z_0 = M^{-1}*r_0, p_0 = z_0$ ;

Step 3. Main loop ...

FOR  $k = 0:\text{MaxIt}$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha*p_k$ ;
- check whether  $x$  is NAN;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- if  $r_{k+1} < r_{\text{best}}$ : save  $x_{k+1}$  as  $x_{\text{best}}$ ;
- 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

safety net check:

- IF  $r_{k+1} > r_{\text{best}}$ 
  1.  $x_{k+1} = x_{\text{best}}$
- END IF

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [spbcgs.c](#) for a safer version

### 10.91.2 Function Documentation

**10.91.2.1** `INT fasp_solver_bdcsr_spbcgs ( block_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  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">block_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>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

03/31/2013

Definition at line 868 of file spbcgs.c.

**10.91.2.2** `INT fasp_solver_dbsr_spbcgs ( 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  $Au=b$  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>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

03/31/2013

Definition at line 479 of file spbcgs.c.

10.91.2.3 INT fasp\_solver\_dcsr\_spbcgs ( 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  $Au=b$  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>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

03/31/2013

Definition at line 90 of file spbcgs.c.

10.91.2.4 INT fasp\_solver\_dstr\_spbcgs ( 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  $Au=b$  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>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

03/31/2013

Definition at line 1257 of file spbcgs.c.

## 10.92 spcg.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient with safety net.

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

## Functions

- [INT fasp\\_solver\\_dcsr\\_spcg](#) ([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  $Au=b$  with safety net.*

- [INT fasp\\_solver\\_bdcsr\\_spcg](#) ([block\\_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  $Au=b$  with safety net.*

- [INT fasp\\_solver\\_dstr\\_spcg](#) ([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  $Au=b$  with safety net.*



### 10.92.1 Detailed Description

Krylov subspace methods – Preconditioned conjugate gradient with safety net.

Abstract algorithm

PCG method to solve  $A*x=b$  is to generate  $\{x_k\}$  to approximate  $x$

Step 0. Given  $A, b, x_0, M$

Step 1. Compute residual  $r_0 = b - A*x_0$  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:  $x_{k+1} = x_k + \alpha*p_k$ ;
- check whether  $x$  is NAN;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- if  $r_{k+1} < r_{best}$ : save  $x_{k+1}$  as  $x_{best}$ ;
- 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

safety net check:

- IF  $r_{\{k+1\}} > r_{\{best\}}$ 
  - 1.  $x_{\{k+1\}} = x_{\{best\}}$
- END IF

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [pcg.c](#) for a version without safety net

### 10.92.2 Function Documentation

**10.92.2.1** `INT fasp_solver_bdcslr_spcg ( block_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  $Au=b$  with safety net.

#### Parameters

<i>A</i>	Pointer to <a href="#">block_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>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

03/28/2013

Definition at line 420 of file spcg.c.

**10.92.2.2** `INT fasp_solver_dcsr_spcg ( 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  $Au=b$  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>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**

03/28/2013

Definition at line 88 of file spcg.c.

**10.92.2.3 INT fasp\_solver\_dstr\_spcg ( 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  $Au=b$  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>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**

03/28/2013

Definition at line 751 of file spcg.c.

## 10.93 spgmres.c File Reference

Krylov subspace methods – Preconditioned GMRes with safety net.

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

### Functions

- **INT fasp\_solver\_dcsr\_spgmres** (**dCSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- **INT fasp\_solver\_bdcsr\_spgmres** (**block\_dCSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- **INT fasp\_solver\_dbsr\_spgmres** (**dBSRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*
- **INT fasp\_solver\_dstr\_spgmres** (**dSTRmat** \*A, **dvector** \*b, **dvector** \*x, **precond** \*pc, const **REAL** tol, const **INT** MaxIt, **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl)  
*Preconditioned GMRES method for solving  $Au=b$  with safe-guard.*

### 10.93.1 Detailed Description

Krylov subspace methods – Preconditioned GMRes with safety net.

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See also [pgmres.c](#) for a variable restarting version.  
See [pgmres.c](#) for a version without safety net

### 10.93.2 Function Documentation

**10.93.2.1 INT fasp\_solver\_bdcsr\_spgmres** ( **block\_dCSRmat** \* A, **dvector** \* b, **dvector** \* x, **precond** \* pc, const **REAL** tol, const **INT** MaxIt, **SHORT** restart, const **SHORT** stop\_type, const **SHORT** prtlvl )

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

#### Parameters

<i>A</i>	Pointer to <a href="#">block_dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</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>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 386 of file spgmres.c.

**10.93.2.2** `INT fasp_solver_dbsr_spgmres ( dBSRmat * A, dvector * b, dvector * x, precondition * pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtlvl )`

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

**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
<i>pc</i>	Pointer to the structure of precondition (precond)
<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 726 of file spgmres.c.

10.93.2.3 `INT fasp_solver_dcsr_spgmres ( dCSRmat * A, dvector * b, dvector * x, precondition * pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtlvl )`

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

## 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>pc</i>	Pointer to the structure of precondition (precond)
<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 Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate

Definition at line 46 of file spgmres.c.

**10.93.2.4** `INT fasp_solver_dstr_spgmres ( dSTRmat * A, dvector * b, dvector * x, precondition * pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtlvl )`

Preconditioned GMRES method for solving  $Au=b$  with safe-guard.

## Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</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>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 1066 of file spgmres.c.

## 10.94 spminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual with safety net.

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

### Functions

- **INT fasp\_solver\_dcsr\_spminres** (dCSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*
- **INT fasp\_solver\_bdcsr\_spminres** (block\_dCSRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*
- **INT fasp\_solver\_dstr\_spminres** (dSTRmat \*A, dvector \*b, dvector \*u, precondition \*pc, const REAL tol, const INT MaxIt, const SHORT stop\_type, const SHORT prtIvl)  
*A preconditioned minimal residual (Minres) method for solving  $Au=b$  with safety net.*

### 10.94.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual with safety net.

Abstract algorithm

Krylov method to solve  $Ax=b$  is to generate  $\{x_k\}$  to approximate  $x$ , where  $x_k$  is the optimal solution in Krylov space

$V_k = \text{span}\{r_0, A*r_0, A^2*r_0, \dots, A^{k-1}*r_0\}$ ,

under some inner product.

For the implementation, we generate a series of  $\{p_k\}$  such that  $V_k = \text{span}\{p_1, \dots, p_k\}$ . Details:

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  $k = 0:\text{MaxIt}$

- get step size  $\alpha = f(r_k, z_k, p_k)$ ;
- update solution:  $x_{k+1} = x_k + \alpha*p_k$ ;
- check whether  $x$  is NAN;
- perform stagnation check;
- update residual:  $r_{k+1} = r_k - \alpha*(A*p_k)$ ;
- if  $r_{k+1} < r_{\text{best}}$ : save  $x_{k+1}$  as  $x_{\text{best}}$ ;
- 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

safety net check:

- IF  $r_{k+1} > r_{\text{best}}$ 
  1.  $x_{k+1} = x_{\text{best}}$
- END IF

#### Note

Refer to Y. Saad 2003 Iterative methods for sparse linear systems (2nd Edition), SIAM  
See [pminres.c](#) for a version without safety net

### 10.94.2 Function Documentation

10.94.2.1 **INT fasp\_solver\_bdcsr\_spminres ( block\_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  $Au=b$  with safety net.

Parameters

A	Pointer to <a href="#">block_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>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 544 of file spminres.c.

**10.94.2.2 INT fasp\_solver\_dcsr\_spminres ( 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  $Au=b$  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>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 95 of file spminres.c.

10.94.2.3 `INT fasp_solver_dstr_spminres ( 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  $Au=b$  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>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 993 of file sminres.c.

## 10.95 spvgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRes with safety net.

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

## Functions

- [INT fasp\\_solver\\_dcsr\\_spvgmres](#) ([dCSRmat](#) \*A, [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 the iteration.
- [INT fasp\\_solver\\_bdcsr\\_spvgmres](#) ([block\\_dCSRmat](#) \*A, [dvector](#) \*b, [dvector](#) \*x, [precond](#) \*pc, const [REAL](#) tol, const [INT](#) MaxIt, [SHORT](#) restart, const [SHORT](#) stop\_type, const [SHORT](#) prtlvl)  
Preconditioned GMRES method for solving  $Au=b$ .
- [INT fasp\\_solver\\_dbsr\\_spvgmres](#) ([dBSRmat](#) \*A, [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 the iteration.
- [INT fasp\\_solver\\_dstr\\_spvgmres](#) ([dSTRmat](#) \*A, [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 the iteration.

### 10.95.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restart GMRes with safety net.

#### Note

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 [pvgmres.c](#) a version without safety net

### 10.95.2 Function Documentation

**10.95.2.1** `INT fasp_solver_bdcsr_spvgmres ( block_dCSRmat * A, dvector * b, dvector * x, precondition * pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtlvl )`

Preconditioned GMRES method for solving  $Au=b$ .

#### Parameters

<i>A</i>	Pointer to <a href="#">block_dCSRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</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>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

04/06/2013

Definition at line 425 of file spvgmres.c.

**10.95.2.2** `INT fasp_solver_dbsr_spvgmres ( dBSRmat * A, 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 the iteration.

## 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
<i>pc</i>	Pointer to the structure of precondition (precond)
<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

04/06/2013

Definition at line 803 of file spvgmres.c.

**10.95.2.3** `INT fasp_solver_dcsr_spvgmres ( dCSRmat * A, 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 the iteration.

## 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>pc</i>	Pointer to the structure of precondition (precond)
<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

04/06/2013 Modified by Chunsheng Feng on 07/22/2013: Add adapt memory allocate

Definition at line 48 of file spvgmres.c.

10.95.2.4 **INT** fasp\_solver\_dstr\_spgmres ( **dSTRmat** \* *A*, **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 the iteration.

#### Parameters

<i>A</i>	Pointer to <a href="#">dSTRmat</a> : the coefficient matrix
<i>b</i>	Pointer to dvector: the right hand side
<i>x</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>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

04/06/2013

Definition at line 1181 of file spvgmres.c.

## 10.96 threads.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](#) (**INT** procid, **INT** nprocs, **INT** n, **INT** \*start, **INT** \*end)  
*Assign Load to each thread.*
- void [fasp\\_set\\_GS\\_threads](#) (**INT** mythreads, **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

### 10.96.1 Detailed Description

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

### 10.96.2 Function Documentation

10.96.2.1 void FASP\_GET\_START\_END ( INT *procid*, INT *nprocs*, 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 83 of file threads.c.

10.96.2.2 void fasp\_set\_GS\_threads ( INT *threads*, 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

TODO: Why put it here??? –Chensong

Definition at line 125 of file threads.c.

### 10.96.3 Variable Documentation

10.96.3.1 INT THDs\_AMG\_GS =0

AMG GS smoothing threads

Definition at line 107 of file threads.c.



### 10.96.3.2 INT THDs\_CPR\_gGS =0

global matrix GS smoothing threads

Definition at line 109 of file threads.c.

### 10.96.3.3 INT THDs\_CPR\_IGS =0

reservoir GS smoothing threads

Definition at line 108 of file threads.c.

## 10.97 timing.c File Reference

Timing subroutines.

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

### Functions

- void [fasp\\_gettime](#) (REAL \*time)  
*Get system time.*

### 10.97.1 Detailed Description

Timing subroutines.

### 10.97.2 Function Documentation

#### 10.97.2.1 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 28 of file timing.c.

## 10.98 vec.c File Reference

Simple operations for vectors.

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

### Functions

- `INT fasp_dvec_isnan (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 (dvector *x, dvector *y)`  
*Copy dvector x to dvector y.*
- `REAL fasp_dvec_maxdiff (dvector *x, dvector *y)`  
*Maximal difference of two dvector x and y.*
- `void fasp_dvec_symdiagscale (dvector *b, dvector *diag)`  
*Symmetric diagonal scaling  $D^{-1/2}b$ .*

### 10.98.1 Detailed Description

Simple operations for vectors.

#### Note

All structures should be initialized before usage.

## 10.98.2 Function Documentation

### 10.98.2.1 void fasp\_dvec\_alloc ( const INT *m*, dvector \* *u* )

Create dvector data space of REAL type.

## Parameters

$m$	Number of rows
$u$	Pointer to dvector (OUTPUT)

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 99 of file vec.c.

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

Copy dvector x to dvector y.

## Parameters

$x$	Pointer to dvector
$y$	Pointer to dvector (MODIFIED)

## Author

Chensong Zhang

## Date

11/16/2009

Definition at line 345 of file vec.c.

10.98.2.3 `dvector fasp_dvec_create ( const INT m )`

Create dvector data space of REAL type.

## Parameters

$m$	Number of rows
-----	----------------

## Returns

u The new dvector

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 56 of file vec.c.

10.98.2.4 void fasp\_dvec\_free ( dvector \* *u* )

Free vector data space of REAL type.

## Parameters

$u$	Pointer to dvector which needs to be deallocated
-----	--

## Author

Chensong Zhang

## Date

2010/04/03

Definition at line 139 of file vec.c.

**10.98.2.5 INT fasp\_dvec\_isnan ( dvector \*  $u$  )**

Check a dvector whether there is NAN.

## Parameters

$u$	Pointer to dvector
-----	--------------------

## Returns

Return TRUE if there is NAN

## Author

Chensong Zhang

## Date

2013/03/31

Definition at line 33 of file vec.c.

**10.98.2.6 REAL fasp\_dvec\_maxdiff ( dvector \*  $x$ , dvector \*  $y$  )**

Maximal difference of two dvector  $x$  and  $y$ .

## Parameters

$x$	Pointer to dvector
$y$	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 368 of file vec.c.

**10.98.2.7 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 vec.c.

**10.98.2.8 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 vec.c.

**10.98.2.9** 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 235 of file vec.c.

**10.98.2.10** void fasp\_dvec\_symdiagscale ( dvector \* *b*, 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 421 of file vec.c.

**10.98.2.11** void fasp\_ivec\_alloc ( const INT *m*, ivector \* *u* )

Create vector data space of INT type.



## Parameters

$m$	Number of rows
$u$	Pointer to ivector (OUTPUT)

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 119 of file vec.c.

**10.98.2.12 ivector fasp\_ivec\_create ( const INT  $m$  )**

Create vector data space of INT type.

## Parameters

$m$	Number of rows
-----	----------------

## Returns

$u$  The new ivector

## Author

Chensong Zhang

## Date

2010/04/06

Definition at line 78 of file vec.c.

**10.98.2.13 void fasp\_ivec\_free ( ivector \*  $u$  )**

Free vector data space of INT type.

## Parameters

$u$	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 vec.c.

10.98.2.14 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 304 of file vec.c.

## 10.99 wrapper.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)*

### 10.99.1 Detailed Description

Wrappers for accessing functions by advanced users.

## 10.99.2 Function Documentation

10.99.2.1 `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 35 of file wrapper.c.

10.99.2.2 `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 85 of file wrapper.c.

**10.99.2.3** `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)

## 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 143 of file wrapper.c.

**10.99.2.4** `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)

## 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 229 of file wrapper.c.



# Index

\_\_FASPBLOCK\_HEADER\_\_  
    fasp\_block.h, [168](#)  
\_\_FASP\_HEADER\_\_  
    fasp.h, [161](#)

## A

    precond\_FASP\_blkoi\_data, [69](#)  
    precond\_sweeping\_data, [72](#)

## A\_diag

    precond\_block\_data, [57](#)

## ABS

    fasp.h, [161](#)

## AMG\_ILU\_levels

    input\_param, [44](#)

## AMG\_Schwarz\_levels

    input\_param, [45](#)

## AMG\_aggregation\_type

    input\_param, [43](#)

## AMG\_aggressive\_level

    input\_param, [43](#)

## AMG\_aggressive\_path

    input\_param, [43](#)

## AMG\_amli\_degree

    input\_param, [43](#)

## AMG\_coarse\_dof

    input\_param, [43](#)

## AMG\_coarse\_scaling

    input\_param, [44](#)

## AMG\_coarse\_solver

    input\_param, [44](#)

## AMG\_coarsening\_type

    input\_param, [44](#)

## AMG\_cycle\_type

    input\_param, [44](#)

## AMG\_data, [23](#)

## AMG\_data\_bsr, [24](#)

## 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, [26](#)

## AMG\_polynomial\_degree

    input\_param, [45](#)

## AMG\_postsmooth\_iter

    input\_param, [45](#)

## AMG\_presmooth\_iter

    input\_param, [45](#)

## AMG\_quality\_bound

    input\_param, [45](#)

## AMG\_relaxation

    input\_param, [45](#)

## AMG\_smooth\_filter

    input\_param, [46](#)

## AMG\_smooth\_order

    input\_param, [46](#)

## AMG\_smoother

    input\_param, [46](#)

## AMG\_strong\_coupled

    input\_param, [46](#)

## AMG\_strong\_threshold

    input\_param, [46](#)

## AMG\_tentative\_smooth

    input\_param, [46](#)

## AMG\_tol

    input\_param, [46](#)

## AMG\_truncation\_threshold

    input\_param, [46](#)

## AMG\_type

    input\_param, [46](#)

## AMLI\_CYCLE

    fasp\_const.h, [172](#)

## ASCEND

    fasp\_const.h, [172](#)

## Abcsr

    precond\_block\_data, [57](#)

## Ai

    precond\_sweeping\_data, [72](#)

## amg.c, [77](#)

    fasp\_solver\_amg, [77](#)

## amg\_setup\_cr.c, [78](#)

    fasp\_amg\_setup\_cr, [78](#)

- amg\_setup\_rs.c, 79
  - fasp\_amg\_setup\_rs, 79
- amg\_setup\_sa.c, 80
  - fasp\_amg\_setup\_sa, 81
  - fasp\_amg\_setup\_sa\_bsr, 81
- amg\_setup\_ua.c, 82
  - fasp\_amg\_setup\_ua, 82
  - fasp\_amg\_setup\_ua\_bsr, 82
- amg\_solve.c, 84
  - fasp\_amg\_solve, 85
  - fasp\_amg\_solve\_amli, 86
  - fasp\_amg\_solve\_nl\_amli, 86
  - fasp\_famg\_solve, 87
- amgparam
  - precond\_block\_data, 57
- amlirecur.c, 87
  - fasp\_amg\_amli\_coef, 88
  - fasp\_solver\_amli, 88
  - fasp\_solver\_nl\_amli, 89
  - fasp\_solver\_nl\_amli\_bsr, 89
- array.c, 90
  - fasp\_array\_cp, 91
  - fasp\_array\_cp\_nc3, 91
  - fasp\_array\_cp\_nc5, 91
  - fasp\_array\_cp\_nc7, 92
  - fasp\_array\_null, 92
  - fasp\_array\_set, 92
  - fasp\_iarray\_cp, 93
  - fasp\_iarray\_set, 93
- BIGREAL
  - fasp\_const.h, 172
- blas\_array.c, 94
  - fasp\_blas\_array\_ax, 95
  - fasp\_blas\_array\_axpby, 96
  - fasp\_blas\_array\_axpy, 96
  - fasp\_blas\_array\_axpyz, 97
  - fasp\_blas\_array\_dotprod, 97
  - fasp\_blas\_array\_norm1, 98
  - fasp\_blas\_array\_norm2, 98
  - fasp\_blas\_array\_norminf, 99
- blas\_bcsr.c, 99
  - fasp\_blas\_bdbsr\_aAxy, 100
  - fasp\_blas\_bdbsr\_mxv, 100
  - fasp\_blas\_bdcsl\_aAxy, 101
  - fasp\_blas\_bdcsl\_mxv, 101
- blas\_bsr.c, 101
  - fasp\_blas\_dbsr\_aAxy, 102
  - fasp\_blas\_dbsr\_aAxy, 103
  - fasp\_blas\_dbsr\_aAxy\_agg, 103
  - fasp\_blas\_dbsr\_axm, 103
  - fasp\_blas\_dbsr\_mxv, 105
  - fasp\_blas\_dbsr\_mxv, 105
  - fasp\_blas\_dbsr\_mxv\_agg, 106
  - fasp\_blas\_dbsr\_rap, 106
  - fasp\_blas\_dbsr\_rap1, 107
  - fasp\_blas\_dbsr\_rap\_agg, 107
- blas\_csr.c, 108
  - fasp\_blas\_dcsr\_aAxy, 109
  - fasp\_blas\_dcsr\_aAxy\_agg, 109
  - fasp\_blas\_dcsr\_add, 110
  - fasp\_blas\_dcsr\_axm, 110
  - fasp\_blas\_dcsr\_bandwidth, 110
  - fasp\_blas\_dcsr\_mxv, 112
  - fasp\_blas\_dcsr\_mxv, 112
  - fasp\_blas\_dcsr\_mxv\_agg, 113
  - fasp\_blas\_dcsr\_ptap, 113
  - fasp\_blas\_dcsr\_rap, 113
  - fasp\_blas\_dcsr\_rap4, 115
  - fasp\_blas\_dcsr\_rap\_agg, 115
  - fasp\_blas\_dcsr\_rap\_agg1, 116
  - fasp\_blas\_dcsr\_vmv, 116
- blas\_csrl.c, 117
  - fasp\_blas\_dcsrl\_mxv, 117
- blas\_smat.c, 118
  - fasp\_blas\_array\_axpy\_nc2, 120
  - fasp\_blas\_array\_axpy\_nc3, 120
  - fasp\_blas\_array\_axpy\_nc5, 120
  - fasp\_blas\_array\_axpy\_nc7, 121
  - fasp\_blas\_array\_axpyz\_nc2, 121
  - fasp\_blas\_array\_axpyz\_nc3, 122
  - fasp\_blas\_array\_axpyz\_nc5, 122
  - fasp\_blas\_array\_axpyz\_nc7, 122
  - fasp\_blas\_smat\_aAxy, 124
  - fasp\_blas\_smat\_add, 124
  - fasp\_blas\_smat\_axm, 125
  - fasp\_blas\_smat\_mul, 125
  - fasp\_blas\_smat\_mul\_nc2, 126
  - fasp\_blas\_smat\_mul\_nc3, 126
  - fasp\_blas\_smat\_mul\_nc5, 126
  - fasp\_blas\_smat\_mul\_nc7, 127
  - fasp\_blas\_smat\_mxv, 127
  - fasp\_blas\_smat\_mxv\_nc2, 127
  - fasp\_blas\_smat\_mxv\_nc3, 129
  - fasp\_blas\_smat\_mxv\_nc5, 129
  - fasp\_blas\_smat\_mxv\_nc7, 129
  - fasp\_blas\_smat\_ymAx, 131
  - fasp\_blas\_smat\_ymAx\_nc2, 131
  - fasp\_blas\_smat\_ymAx\_nc3, 132
  - fasp\_blas\_smat\_ymAx\_nc5, 132
  - fasp\_blas\_smat\_ymAx\_nc7, 132
  - fasp\_blas\_smat\_ymAx\_ns, 133
  - fasp\_blas\_smat\_ymAx\_ns2, 133
  - fasp\_blas\_smat\_ymAx\_ns3, 134
  - fasp\_blas\_smat\_ymAx\_ns5, 134
  - fasp\_blas\_smat\_ymAx\_ns7, 135
  - fasp\_blas\_smat\_ypAx, 135
  - fasp\_blas\_smat\_ypAx\_nc2, 136



- fasp\_blas\_smat\_yPAx\_nc3, [136](#)
  - fasp\_blas\_smat\_yPAx\_nc5, [136](#)
  - fasp\_blas\_smat\_yPAx\_nc7, [138](#)
- blas\_str.c, [138](#)
  - fasp\_blas\_dstr\_aApy, [139](#)
  - fasp\_blas\_dstr\_mxv, [139](#)
  - fasp\_dstr\_diagscale, [139](#)
- blas\_vec.c, [140](#)
  - fasp\_blas\_dvec\_axpy, [141](#)
  - fasp\_blas\_dvec\_axpyz, [141](#)
  - fasp\_blas\_dvec\_dotprod, [141](#)
  - fasp\_blas\_dvec\_norm1, [142](#)
  - fasp\_blas\_dvec\_norm2, [142](#)
  - fasp\_blas\_dvec\_norminf, [143](#)
  - fasp\_blas\_dvec\_relerr, [143](#)
- block\_BSR, [28](#)
  - fasp\_block.h, [168](#)
- block\_Reservoir, [31](#)
  - fasp\_block.h, [168](#)
- block\_dCSRmat, [28](#)
  - fasp\_block.h, [168](#)
- block\_dvector, [29](#)
  - fasp\_block.h, [168](#)
- block\_iCSRmat, [29](#)
  - fasp\_block.h, [168](#)
- block\_ivector, [30](#)
  - fasp\_block.h, [168](#)
- CF\_ORDER
  - fasp\_const.h, [173](#)
- CGPT
  - fasp\_const.h, [173](#)
- CLASSIC\_AMG
  - fasp\_const.h, [173](#)
- COARSE\_AC
  - fasp\_const.h, [173](#)
- COARSE\_CR
  - fasp\_const.h, [173](#)
- COARSE\_MIS
  - fasp\_const.h, [173](#)
- COARSE\_RS
  - fasp\_const.h, [173](#)
- COARSE\_RSP
  - fasp\_const.h, [173](#)
- CPFIRST
  - fasp\_const.h, [174](#)
- checkmat.c, [144](#)
  - fasp\_check\_dCSRmat, [145](#)
  - fasp\_check\_diagdom, [145](#)
  - fasp\_check\_diagpos, [145](#)
  - fasp\_check\_diagzero, [147](#)
  - fasp\_check\_iCSRmat, [147](#)
  - fasp\_check\_symm, [147](#)
- coarsening\_cr.c, [149](#)
  - fasp\_amg\_coarsening\_cr, [149](#)
- coarsening\_rs.c, [150](#)
  - fasp\_amg\_coarsening\_rs, [151](#)
- convert.c, [152](#)
  - endian\_convert\_int, [153](#)
  - endian\_convert\_real, [153](#)
  - fasp\_aux\_bbyteToldouble, [153](#)
  - fasp\_aux\_change\_endian4, [155](#)
  - fasp\_aux\_change\_endian8, [155](#)
- count
  - fasp.h, [165](#)
- dBSRmat, [31](#)
  - fasp\_block.h, [168](#)
  - JA, [32](#)
  - val, [32](#)
- dCOOmat, [32](#)
  - fasp.h, [164](#)
- dCSRmat, [33](#)
  - fasp.h, [164](#)
- dCSRmat, [34](#)
  - fasp.h, [164](#)
- dCSRmat2SAMGInput
  - interface\_samg.c, [216](#)
- DESCEND
  - fasp\_const.h, [174](#)
- DIAGONAL\_PREF
  - fasp.h, [161](#)
- DLMALLOC
  - fasp.h, [161](#)
- dSTRmat, [35](#)
  - fasp.h, [164](#)
- ddenmat, [34](#)
  - fasp.h, [164](#)
- diag
  - precond\_block\_reservoir\_data, [59](#)
- diaginv
  - precond\_FASP\_blkoi\_data, [69](#)
  - precond\_block\_reservoir\_data, [59](#)
- diaginv\_S
  - precond\_FASP\_blkoi\_data, [69](#)
- diaginv\_noscale
  - precond\_FASP\_blkoi\_data, [69](#)
- diaginvS
  - precond\_block\_reservoir\_data, [60](#)
- dlength
  - io.c, [243](#)
- doxygen.h, [156](#)
- dvector, [36](#)
  - fasp.h, [164](#)
- dvector2SAMGInput
  - interface\_samg.c, [217](#)
- e
  - grid2d, [37](#)

ERROR\_ALLOC\_MEM  
     fasp\_const.h, 174  
 ERROR\_AMG\_COARSE\_TYPE  
     fasp\_const.h, 174  
 ERROR\_AMG\_COARSEING  
     fasp\_const.h, 174  
 ERROR\_AMG\_INTERP\_TYPE  
     fasp\_const.h, 174  
 ERROR\_AMG\_SMOOTH\_TYPE  
     fasp\_const.h, 174  
 ERROR\_DATA\_STRUCTURE  
     fasp\_const.h, 174  
 ERROR\_DATA\_ZERODIAG  
     fasp\_const.h, 174  
 ERROR\_DUMMY\_VAR  
     fasp\_const.h, 175  
 ERROR\_INPUT\_PAR  
     fasp\_const.h, 175  
 ERROR\_LIC\_TYPE  
     fasp\_const.h, 175  
 ERROR\_MAT\_SIZE  
     fasp\_const.h, 175  
 ERROR\_MISC  
     fasp\_const.h, 175  
 ERROR\_NUM\_BLOCKS  
     fasp\_const.h, 175  
 ERROR\_OPEN\_FILE  
     fasp\_const.h, 175  
 ERROR\_QUAD\_DIM  
     fasp\_const.h, 175  
 ERROR\_QUAD\_TYPE  
     fasp\_const.h, 175  
 ERROR\_REGRESS  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_EXIT  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_ILUSETUP  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_MAXIT  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_MISC  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_PRECTYPE  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_SOLSTAG  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_STAG  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_TOLSMALL  
     fasp\_const.h, 176  
 ERROR\_SOLVER\_TYPE  
     fasp\_const.h, 177  
 ERROR\_UNKNOWN  
     fasp\_const.h, 177  
 ERROR\_WRONG\_FILE  
     fasp\_const.h, 177  
 edges  
     grid2d, 37  
 ediri  
     grid2d, 37  
 efather  
     grid2d, 37  
 eigen.c, 156  
     fasp\_dcsr\_eig, 156  
 endian\_convert\_int  
     convert.c, 153  
 endian\_convert\_real  
     convert.c, 153  
 FALSE  
     fasp\_const.h, 177  
 FASP\_GET\_START\_END  
     threads.c, 454  
 FASP\_GSRB  
     fasp.h, 161  
 FASP\_SUCCESS  
     fasp\_const.h, 177  
 FASP\_USE\_ILU  
     fasp.h, 161  
 FASP\_VERSION  
     fasp.h, 161  
 FGPT  
     fasp\_const.h, 177  
 FPFIRST  
     fasp\_const.h, 177  
 famg.c, 157  
     fasp\_solver\_famg, 157  
 fasp.h, 158  
     \_\_FASP\_HEADER\_\_, 161  
     ABS, 161  
     count, 165  
     dCOOmat, 164  
     dCSRLmat, 164  
     dCSRmat, 164  
     DIAGONAL\_PREF, 161  
     DLMALLOC, 161  
     dSTRmat, 164  
     ddenmat, 164  
     dvector, 164  
     FASP\_GSRB, 161  
     FASP\_USE\_ILU, 161  
     FASP\_VERSION, 161  
     GE, 162  
     GT, 162  
     grid2d, 164  
     iCOOmat, 164  
     iCSRmat, 164  
     IMAP, 165

- INT, [162](#)
- ISNAN, [162](#)
- idenmat, [165](#)
- ivector, [165](#)
- LE, [162](#)
- LONG, [162](#)
- LONGLONG, [162](#)
- LS, [162](#)
- LinkedList, [165](#)
- ListElement, [165](#)
- MAX, [163](#)
- MAXIMAP, [165](#)
- MIN, [163](#)
- NEDMALLOC, [163](#)
- nx\_rb, [165](#)
- ny\_rb, [166](#)
- nz\_rb, [166](#)
- PUT\_INT, [163](#)
- PUT\_REAL, [163](#)
- pcgrid2d, [165](#)
- pgrid2d, [165](#)
- REAL, [163](#)
- RS\_C1, [163](#)
- SHORT, [164](#)
- total\_alloc\_count, [166](#)
- total\_alloc\_mem, [166](#)
- fasp\_BinarySearch
  - ordering.c, [280](#)
- fasp\_Schwarz\_data\_free
  - init.c, [212](#)
- fasp\_Schwarz\_get\_block\_matrix
  - schwarz\_setup.c, [358](#)
- fasp\_Schwarz\_setup
  - schwarz\_setup.c, [358](#)
- fasp\_amg\_amli\_coef
  - amlirecur.c, [88](#)
- fasp\_amg\_coarsening\_cr
  - coarsening\_cr.c, [149](#)
- fasp\_amg\_coarsening\_rs
  - coarsening\_rs.c, [151](#)
- fasp\_amg\_data\_bsr\_create
  - init.c, [209](#)
- fasp\_amg\_data\_bsr\_free
  - init.c, [209](#)
- fasp\_amg\_data\_create
  - init.c, [210](#)
- fasp\_amg\_data\_free
  - init.c, [210](#)
- fasp\_amg\_interp
  - interpolation.c, [219](#)
- fasp\_amg\_interp1
  - interpolation.c, [220](#)
- fasp\_amg\_interp\_em
  - interpolation\_em.c, [221](#)
- fasp\_amg\_interp\_trunc
  - interpolation.c, [220](#)
- fasp\_amg\_setup\_cr
  - amg\_setup\_cr.c, [78](#)
- fasp\_amg\_setup\_rs
  - amg\_setup\_rs.c, [79](#)
- fasp\_amg\_setup\_sa
  - amg\_setup\_sa.c, [81](#)
- fasp\_amg\_setup\_sa\_bsr
  - amg\_setup\_sa.c, [81](#)
- fasp\_amg\_setup\_ua
  - amg\_setup\_ua.c, [82](#)
- fasp\_amg\_setup\_ua\_bsr
  - amg\_setup\_ua.c, [82](#)
- fasp\_amg\_solve
  - amg\_solve.c, [85](#)
- fasp\_amg\_solve\_amli
  - amg\_solve.c, [86](#)
- fasp\_amg\_solve\_nl\_amli
  - amg\_solve.c, [86](#)
- fasp\_array\_cp
  - array.c, [91](#)
- fasp\_array\_cp\_nc3
  - array.c, [91](#)
- fasp\_array\_cp\_nc5
  - array.c, [91](#)
- fasp\_array\_cp\_nc7
  - array.c, [92](#)
- fasp\_array\_null
  - array.c, [92](#)
- fasp\_array\_set
  - array.c, [92](#)
- fasp\_aux\_bbyteToldouble
  - convert.c, [153](#)
- fasp\_aux\_change\_endian4
  - convert.c, [155](#)
- fasp\_aux\_change\_endian8
  - convert.c, [155](#)
- fasp\_aux\_dQuickSort
  - ordering.c, [276](#)
- fasp\_aux\_dQuickSortIndex
  - ordering.c, [276](#)
- fasp\_aux\_givens
  - givens.c, [195](#)
- fasp\_aux\_iQuickSort
  - ordering.c, [276](#)
- fasp\_aux\_iQuickSortIndex
  - ordering.c, [278](#)
- fasp\_aux\_merge
  - ordering.c, [278](#)
- fasp\_aux\_msor
  - ordering.c, [279](#)
- fasp\_aux\_unique
  - ordering.c, [279](#)

fasp\_bdcsr\_free  
     sparse\_block.c, [394](#)  
 fasp\_blas\_array\_ax  
     blas\_array.c, [95](#)  
 fasp\_blas\_array\_axpby  
     blas\_array.c, [96](#)  
 fasp\_blas\_array\_axpy  
     blas\_array.c, [96](#)  
 fasp\_blas\_array\_axpy\_nc2  
     blas\_smat.c, [120](#)  
 fasp\_blas\_array\_axpy\_nc3  
     blas\_smat.c, [120](#)  
 fasp\_blas\_array\_axpy\_nc5  
     blas\_smat.c, [120](#)  
 fasp\_blas\_array\_axpy\_nc7  
     blas\_smat.c, [121](#)  
 fasp\_blas\_array\_axpyz  
     blas\_array.c, [97](#)  
 fasp\_blas\_array\_axpyz\_nc2  
     blas\_smat.c, [121](#)  
 fasp\_blas\_array\_axpyz\_nc3  
     blas\_smat.c, [122](#)  
 fasp\_blas\_array\_axpyz\_nc5  
     blas\_smat.c, [122](#)  
 fasp\_blas\_array\_axpyz\_nc7  
     blas\_smat.c, [122](#)  
 fasp\_blas\_array\_dotprod  
     blas\_array.c, [97](#)  
 fasp\_blas\_array\_norm1  
     blas\_array.c, [98](#)  
 fasp\_blas\_array\_norm2  
     blas\_array.c, [98](#)  
 fasp\_blas\_array\_norminf  
     blas\_array.c, [99](#)  
 fasp\_blas\_bdbsr\_aAxy  
     blas\_bcsr.c, [100](#)  
 fasp\_blas\_bdbsr\_mxv  
     blas\_bcsr.c, [100](#)  
 fasp\_blas\_bdcsr\_aAxy  
     blas\_bcsr.c, [101](#)  
 fasp\_blas\_bdcsr\_mxv  
     blas\_bcsr.c, [101](#)  
 fasp\_blas\_dbsr\_aAxpby  
     blas\_bsr.c, [102](#)  
 fasp\_blas\_dbsr\_aAxy  
     blas\_bsr.c, [103](#)  
 fasp\_blas\_dbsr\_aAxy\_agg  
     blas\_bsr.c, [103](#)  
 fasp\_blas\_dbsr\_axm  
     blas\_bsr.c, [103](#)  
 fasp\_blas\_dbsr\_axm  
     blas\_bsr.c, [105](#)  
 fasp\_blas\_dbsr\_mxv  
     blas\_bsr.c, [105](#)  
 fasp\_blas\_dbsr\_mxv\_agg  
     blas\_bsr.c, [106](#)  
 fasp\_blas\_dbsr\_rap  
     blas\_bsr.c, [106](#)  
 fasp\_blas\_dbsr\_rap1  
     blas\_bsr.c, [107](#)  
 fasp\_blas\_dbsr\_rap\_agg  
     blas\_bsr.c, [107](#)  
 fasp\_blas\_dcsr\_aAxy  
     blas\_csr.c, [109](#)  
 fasp\_blas\_dcsr\_aAxy\_agg  
     blas\_csr.c, [109](#)  
 fasp\_blas\_dcsr\_add  
     blas\_csr.c, [110](#)  
 fasp\_blas\_dcsr\_axm  
     blas\_csr.c, [110](#)  
 fasp\_blas\_dcsr\_bandwidth  
     blas\_csr.c, [110](#)  
 fasp\_blas\_dcsr\_mxm  
     blas\_csr.c, [112](#)  
 fasp\_blas\_dcsr\_mxv  
     blas\_csr.c, [112](#)  
 fasp\_blas\_dcsr\_mxv\_agg  
     blas\_csr.c, [113](#)  
 fasp\_blas\_dcsr\_ptap  
     blas\_csr.c, [113](#)  
 fasp\_blas\_dcsr\_rap  
     blas\_csr.c, [113](#)  
 fasp\_blas\_dcsr\_rap2  
     rap.c, [356](#)  
 fasp\_blas\_dcsr\_rap4  
     blas\_csr.c, [115](#)  
 fasp\_blas\_dcsr\_rap\_agg  
     blas\_csr.c, [115](#)  
 fasp\_blas\_dcsr\_rap\_agg1  
     blas\_csr.c, [116](#)  
 fasp\_blas\_dcsr\_vmv  
     blas\_csr.c, [116](#)  
 fasp\_blas\_dcsr\_l\_mxv  
     blas\_csr.c, [117](#)  
 fasp\_blas\_dstr\_aAxy  
     blas\_str.c, [139](#)  
 fasp\_blas\_dstr\_mxv  
     blas\_str.c, [139](#)  
 fasp\_blas\_dvec\_axpy  
     blas\_vec.c, [141](#)  
 fasp\_blas\_dvec\_axpyz  
     blas\_vec.c, [141](#)  
 fasp\_blas\_dvec\_dotprod  
     blas\_vec.c, [141](#)  
 fasp\_blas\_dvec\_norm1  
     blas\_vec.c, [142](#)  
 fasp\_blas\_dvec\_norm2  
     blas\_vec.c, [142](#)

fasp\_blas\_dvec\_norminf  
blas\_vec.c, [143](#)

fasp\_blas\_dvec\_relerr  
blas\_vec.c, [143](#)

fasp\_blas\_smat\_Linfinity  
smat.c, [362](#)

fasp\_blas\_smat\_aAxpby  
blas\_smat.c, [124](#)

fasp\_blas\_smat\_add  
blas\_smat.c, [124](#)

fasp\_blas\_smat\_axm  
blas\_smat.c, [125](#)

fasp\_blas\_smat\_inv  
smat.c, [360](#)

fasp\_blas\_smat\_inv\_nc  
smat.c, [360](#)

fasp\_blas\_smat\_inv\_nc2  
smat.c, [360](#)

fasp\_blas\_smat\_inv\_nc3  
smat.c, [361](#)

fasp\_blas\_smat\_inv\_nc4  
smat.c, [361](#)

fasp\_blas\_smat\_inv\_nc5  
smat.c, [361](#)

fasp\_blas\_smat\_inv\_nc7  
smat.c, [362](#)

fasp\_blas\_smat\_invp\_nc  
smat.c, [362](#)

fasp\_blas\_smat\_mul  
blas\_smat.c, [125](#)

fasp\_blas\_smat\_mul\_nc2  
blas\_smat.c, [126](#)

fasp\_blas\_smat\_mul\_nc3  
blas\_smat.c, [126](#)

fasp\_blas\_smat\_mul\_nc5  
blas\_smat.c, [126](#)

fasp\_blas\_smat\_mul\_nc7  
blas\_smat.c, [127](#)

fasp\_blas\_smat\_mxv  
blas\_smat.c, [127](#)

fasp\_blas\_smat\_mxv\_nc2  
blas\_smat.c, [127](#)

fasp\_blas\_smat\_mxv\_nc3  
blas\_smat.c, [129](#)

fasp\_blas\_smat\_mxv\_nc5  
blas\_smat.c, [129](#)

fasp\_blas\_smat\_mxv\_nc7  
blas\_smat.c, [129](#)

fasp\_blas\_smat\_ymAx  
blas\_smat.c, [131](#)

fasp\_blas\_smat\_ymAx\_nc2  
blas\_smat.c, [131](#)

fasp\_blas\_smat\_ymAx\_nc3  
blas\_smat.c, [132](#)

fasp\_blas\_smat\_ymAx\_nc5  
blas\_smat.c, [132](#)

fasp\_blas\_smat\_ymAx\_nc7  
blas\_smat.c, [132](#)

fasp\_blas\_smat\_ymAx\_ns  
blas\_smat.c, [133](#)

fasp\_blas\_smat\_ymAx\_ns2  
blas\_smat.c, [133](#)

fasp\_blas\_smat\_ymAx\_ns3  
blas\_smat.c, [134](#)

fasp\_blas\_smat\_ymAx\_ns5  
blas\_smat.c, [134](#)

fasp\_blas\_smat\_ymAx\_ns7  
blas\_smat.c, [135](#)

fasp\_blas\_smat\_ypAx  
blas\_smat.c, [135](#)

fasp\_blas\_smat\_ypAx\_nc2  
blas\_smat.c, [136](#)

fasp\_blas\_smat\_ypAx\_nc3  
blas\_smat.c, [136](#)

fasp\_blas\_smat\_ypAx\_nc5  
blas\_smat.c, [136](#)

fasp\_blas\_smat\_ypAx\_nc7  
blas\_smat.c, [138](#)

fasp\_block.h, [166](#)

\_\_FASPBLOCK\_HEADER\_\_, [168](#)

block\_BSR, [168](#)

block\_Reservoir, [168](#)

block\_dCSRmat, [168](#)

block\_dvector, [168](#)

block\_iCSRmat, [168](#)

block\_ivector, [168](#)

dBSRmat, [168](#)

precond\_block\_reservoir\_data, [169](#)

SMOOTHER\_BLKOIL, [168](#)

SMOOTHER\_SPETEN, [168](#)

fasp\_check\_dCSRmat  
checkmat.c, [145](#)

fasp\_check\_diagdom  
checkmat.c, [145](#)

fasp\_check\_diagpos  
checkmat.c, [145](#)

fasp\_check\_diagzero  
checkmat.c, [147](#)

fasp\_check\_iCSRmat  
checkmat.c, [147](#)

fasp\_check\_symm  
checkmat.c, [147](#)

fasp\_chkerr  
message.c, [270](#)

fasp\_const.h, [169](#)

AMLI\_CYCLE, [172](#)

ASCEND, [172](#)

BIGREAL, [172](#)

CF\_ORDER, 173  
CGPT, 173  
CLASSIC\_AMG, 173  
COARSE\_AC, 173  
COARSE\_CR, 173  
COARSE\_MIS, 173  
COARSE\_RS, 173  
COARSE\_RSP, 173  
CPFIRST, 174  
DESCEND, 174  
ERROR\_ALLOC\_MEM, 174  
ERROR\_AMG\_COARSE\_TYPE, 174  
ERROR\_AMG\_COARSEING, 174  
ERROR\_AMG\_INTERP\_TYPE, 174  
ERROR\_AMG\_SMOOTH\_TYPE, 174  
ERROR\_DATA\_STRUCTURE, 174  
ERROR\_DATA\_ZERODIAG, 174  
ERROR\_DUMMY\_VAR, 175  
ERROR\_INPUT\_PAR, 175  
ERROR\_LIC\_TYPE, 175  
ERROR\_MAT\_SIZE, 175  
ERROR\_MISC, 175  
ERROR\_NUM\_BLOCKS, 175  
ERROR\_OPEN\_FILE, 175  
ERROR\_QUAD\_DIM, 175  
ERROR\_QUAD\_TYPE, 175  
ERROR\_REGRESS, 176  
ERROR\_SOLVER\_EXIT, 176  
ERROR\_SOLVER\_ILUSETUP, 176  
ERROR\_SOLVER\_MAXIT, 176  
ERROR\_SOLVER\_MISC, 176  
ERROR\_SOLVER\_PRECTYPE, 176  
ERROR\_SOLVER\_SOLSTAG, 176  
ERROR\_SOLVER\_STAG, 176  
ERROR\_SOLVER\_TOLSMALL, 176  
ERROR\_SOLVER\_TYPE, 177  
ERROR\_UNKNOWN, 177  
ERROR\_WRONG\_FILE, 177  
FALSE, 177  
FASP\_SUCCESS, 177  
FGPT, 177  
FPFIRST, 177  
G0PT, 177  
ILUk, 178  
ILUt, 178  
ILUtp, 178  
INTERP\_DIR, 178  
INTERP\_ENG, 178  
INTERP\_STD, 178  
ISPT, 178  
MAT\_BSR, 179  
MAT\_CSR, 179  
MAT\_CSRL, 179  
MAT\_FREE, 179  
MAT\_STR, 179  
MAT\_SymCSR, 179  
MAT\_bBSR, 178  
MAT\_bCSR, 179  
MAX\_AMG\_LVL, 179  
MAX\_CRATE, 180  
MAX\_REFINE\_LVL, 180  
MAX\_RESTART, 180  
MAX\_STAG, 180  
MIN\_CDOF, 180  
MIN\_CRATE, 180  
NL\_AMLI\_CYCLE, 180  
NO\_ORDER, 180  
OFF, 181  
ON, 181  
OPENMP\_HOLDS, 181  
PAIRWISE, 181  
PREC\_AMG, 181  
PREC\_DIAG, 181  
PREC\_FMG, 181  
PREC\_ILU, 181  
PREC\_NULL, 182  
PREC\_SCHWARZ, 182  
PRINT\_ALL, 182  
PRINT\_MIN, 182  
PRINT\_MORE, 182  
PRINT\_MOST, 182  
PRINT\_NONE, 182  
PRINT\_SOME, 182  
SA\_AMG, 182  
SCHWARZ\_BACKWARD, 183  
SCHWARZ\_FORWARD, 183  
SCHWARZ\_SYMMETRIC, 183  
SMALLREAL, 183  
SMALLREAL2, 183  
SMOOTHER\_CG, 183  
SMOOTHER\_GS, 183  
SMOOTHER\_GSOR, 183  
SMOOTHER\_JACOBI, 184  
SMOOTHER\_L1DIAG, 184  
SMOOTHER\_POLY, 184  
SMOOTHER\_SGS, 184  
SMOOTHER\_SGSOR, 184  
SMOOTHER\_SOR, 184  
SMOOTHER\_SSOR, 184  
SOLVER\_AMG, 184  
SOLVER\_BiCGstab, 185  
SOLVER\_CG, 185  
SOLVER\_DEFAULT, 185  
SOLVER\_FMG, 185  
SOLVER\_GCG, 185  
SOLVER\_GCR, 185  
SOLVER\_GMRES, 185  
SOLVER\_MUMPS, 186

- SOLVER\_MinRes, [185](#)
- SOLVER\_SBiCGstab, [186](#)
- SOLVER\_SCG, [186](#)
- SOLVER\_SGCG, [186](#)
- SOLVER\_SGMRES, [186](#)
- SOLVER\_SMinRes, [186](#)
- SOLVER\_SUPERLU, [186](#)
- SOLVER\_SVFGMRES, [186](#)
- SOLVER\_SVGMRES, [186](#)
- SOLVER\_UMFPACK, [187](#)
- SOLVER\_VFGMRES, [187](#)
- SOLVER\_VGMRES, [187](#)
- STAG\_RATIO, [187](#)
- STOP\_MOD\_REL\_RES, [187](#)
- STOP\_REL\_PRECRES, [187](#)
- STOP\_REL\_RES, [187](#)
- TRUE, [187](#)
- UA\_AMG, [188](#)
- UNPT, [188](#)
- USERDEFINED, [188](#)
- V\_CYCLE, [188](#)
- VMB, [188](#)
- W\_CYCLE, [188](#)
- fasp\_dbsr\_Linfinity\_dcsr
  - sparse\_block.c, [396](#)
- fasp\_dbsr\_alloc
  - sparse\_bsr.c, [398](#)
- fasp\_dbsr\_cp
  - sparse\_bsr.c, [398](#)
- fasp\_dbsr\_create
  - sparse\_bsr.c, [399](#)
- fasp\_dbsr\_diagLU
  - sparse\_bsr.c, [401](#)
- fasp\_dbsr\_diagLU2
  - sparse\_bsr.c, [402](#)
- fasp\_dbsr\_diaginv
  - sparse\_bsr.c, [399](#)
- fasp\_dbsr\_diaginv2
  - sparse\_bsr.c, [400](#)
- fasp\_dbsr\_diaginv3
  - sparse\_bsr.c, [400](#)
- fasp\_dbsr\_diaginv4
  - sparse\_bsr.c, [401](#)
- fasp\_dbsr\_diagpref
  - sparse\_bsr.c, [402](#)
- fasp\_dbsr\_free
  - sparse\_bsr.c, [403](#)
- fasp\_dbsr\_getblk
  - sparse\_block.c, [395](#)
- fasp\_dbsr\_getblk\_dcsr
  - sparse\_block.c, [395](#)
- fasp\_dbsr\_getdiag
  - sparse\_bsr.c, [403](#)
- fasp\_dbsr\_getdiaginv
  - sparse\_bsr.c, [404](#)
- fasp\_dbsr\_null
  - sparse\_bsr.c, [404](#)
- fasp\_dbsr\_plot
  - graphics.c, [202](#)
- fasp\_dbsr\_print
  - io.c, [224](#)
- fasp\_dbsr\_read
  - io.c, [224](#)
- fasp\_dbsr\_subplot
  - graphics.c, [203](#)
- fasp\_dbsr\_trans
  - sparse\_bsr.c, [404](#)
- fasp\_dbsr\_write
  - io.c, [225](#)
- fasp\_dbsr\_write\_coo
  - io.c, [225](#)
- fasp\_dcoo1\_read
  - io.c, [226](#)
- fasp\_dcoo\_alloc
  - sparse\_coo.c, [405](#)
- fasp\_dcoo\_create
  - sparse\_coo.c, [406](#)
- fasp\_dcoo\_free
  - sparse\_coo.c, [406](#)
- fasp\_dcoo\_print
  - io.c, [226](#)
- fasp\_dcoo\_read
  - io.c, [227](#)
- fasp\_dcoo\_shift
  - sparse\_coo.c, [406](#)
- fasp\_dcoo\_shift\_read
  - io.c, [227](#)
- fasp\_dcoo\_write
  - io.c, [228](#)
- fasp\_dcsr\_CMK\_order
  - ordering.c, [280](#)
- fasp\_dcsr\_RCMK\_order
  - ordering.c, [281](#)
- fasp\_dcsr\_Schwarz\_backward\_smoother
  - schwarz\_setup.c, [357](#)
- fasp\_dcsr\_Schwarz\_forward\_smoother
  - schwarz\_setup.c, [357](#)
- fasp\_dcsr\_alloc
  - sparse\_csr.c, [408](#)
- fasp\_dcsr\_compress
  - sparse\_csr.c, [409](#)
- fasp\_dcsr\_compress\_inplace
  - sparse\_csr.c, [409](#)
- fasp\_dcsr\_cp
  - sparse\_csr.c, [410](#)
- fasp\_dcsr\_create
  - sparse\_csr.c, [410](#)
- fasp\_dcsr\_diagpref

sparse\_csr.c, 410  
 fasp\_dcsr\_eig  
   eigen.c, 156  
 fasp\_dcsr\_free  
   sparse\_csr.c, 412  
 fasp\_dcsr\_getblk  
   sparse\_block.c, 396  
 fasp\_dcsr\_getcol  
   sparse\_csr.c, 412  
 fasp\_dcsr\_getdiag  
   sparse\_csr.c, 413  
 fasp\_dcsr\_multicoloring  
   sparse\_csr.c, 413  
 fasp\_dcsr\_null  
   sparse\_csr.c, 414  
 fasp\_dcsr\_perm  
   sparse\_csr.c, 414  
 fasp\_dcsr\_permz  
   sparse\_csr.c, 414  
 fasp\_dcsr\_plot  
   graphics.c, 203  
 fasp\_dcsr\_print  
   io.c, 228  
 fasp\_dcsr\_read  
   io.c, 229  
 fasp\_dcsr\_regdiag  
   sparse\_csr.c, 415  
 fasp\_dcsr\_shift  
   sparse\_csr.c, 415  
 fasp\_dcsr\_sort  
   sparse\_csr.c, 416  
 fasp\_dcsr\_sortz  
   sparse\_csr.c, 416  
 fasp\_dcsr\_subplot  
   graphics.c, 204  
 fasp\_dcsr\_symdiagscale  
   sparse\_csr.c, 416  
 fasp\_dcsr\_sympat  
   sparse\_csr.c, 417  
 fasp\_dcsr\_trans  
   sparse\_csr.c, 417  
 fasp\_dcsr\_transz  
   sparse\_csr.c, 418  
 fasp\_dcsr\_write\_coo  
   io.c, 229  
 fasp\_dcsl\_create  
   sparse\_csr.c, 422  
 fasp\_dcsl\_free  
   sparse\_csr.c, 422  
 fasp\_dcsrvec1\_read  
   io.c, 229  
 fasp\_dcsrvec1\_write  
   io.c, 230  
 fasp\_dcsrvec2\_read  
   io.c, 231  
 fasp\_dcsrvec2\_write  
   io.c, 231  
 fasp\_dmtx\_read  
   io.c, 232  
 fasp\_dmtxsym\_read  
   io.c, 232  
 fasp\_dstr\_alloc  
   sparse\_str.c, 423  
 fasp\_dstr\_cp  
   sparse\_str.c, 423  
 fasp\_dstr\_create  
   sparse\_str.c, 425  
 fasp\_dstr\_diagscale  
   blas\_str.c, 139  
 fasp\_dstr\_free  
   sparse\_str.c, 425  
 fasp\_dstr\_null  
   sparse\_str.c, 426  
 fasp\_dstr\_print  
   io.c, 234  
 fasp\_dstr\_read  
   io.c, 234  
 fasp\_dstr\_write  
   io.c, 235  
 fasp\_dvec\_alloc  
   vec.c, 457  
 fasp\_dvec\_cp  
   vec.c, 458  
 fasp\_dvec\_create  
   vec.c, 458  
 fasp\_dvec\_free  
   vec.c, 458  
 fasp\_dvec\_isnan  
   vec.c, 460  
 fasp\_dvec\_maxdiff  
   vec.c, 460  
 fasp\_dvec\_null  
   vec.c, 461  
 fasp\_dvec\_print  
   io.c, 235  
 fasp\_dvec\_rand  
   vec.c, 461  
 fasp\_dvec\_read  
   io.c, 236  
 fasp\_dvec\_set  
   vec.c, 462  
 fasp\_dvec\_symdiagscale  
   vec.c, 462  
 fasp\_dvec\_write  
   io.c, 236  
 fasp\_dvecind\_read  
   io.c, 236  
 fasp\_dvecind\_write



io.c, [237](#)  
fasp\_famg\_solve  
  amg\_solve.c, [87](#)  
fasp\_format\_bdcscr\_dcsr  
  formats.c, [190](#)  
fasp\_format\_dbsr\_dcoo  
  formats.c, [190](#)  
fasp\_format\_dbsr\_dcsr  
  formats.c, [191](#)  
fasp\_format\_dcoo\_dcsr  
  formats.c, [191](#)  
fasp\_format\_dcsr\_dbsr  
  formats.c, [192](#)  
fasp\_format\_dcsr\_dcoo  
  formats.c, [192](#)  
fasp\_format\_dcsrl\_dcsr  
  formats.c, [193](#)  
fasp\_format\_dstr\_dbsr  
  formats.c, [193](#)  
fasp\_format\_dstr\_dcsr  
  formats.c, [194](#)  
fasp\_fwrapper\_amg\_  
  wrapper.c, [465](#)  
fasp\_fwrapper\_krylov\_amg\_  
  wrapper.c, [465](#)  
fasp\_gauss2d  
  quadrature.c, [354](#)  
fasp\_generate\_diaginv\_block  
  smoother\_str.c, [386](#)  
fasp\_gettime  
  timing.c, [455](#)  
fasp\_grid2d\_plot  
  graphics.c, [204](#)  
fasp\_hb\_read  
  io.c, [237](#)  
fasp\_iarray\_cp  
  array.c, [93](#)  
fasp\_iarray\_set  
  array.c, [93](#)  
fasp\_icsr\_cp  
  sparse\_csr.c, [418](#)  
fasp\_icsr\_create  
  sparse\_csr.c, [419](#)  
fasp\_icsr\_free  
  sparse\_csr.c, [419](#)  
fasp\_icsr\_null  
  sparse\_csr.c, [419](#)  
fasp\_icsr\_trans  
  sparse\_csr.c, [421](#)  
fasp\_iden\_free  
  smat.c, [363](#)  
fasp\_ilu\_data\_alloc  
  init.c, [211](#)  
fasp\_ilu\_data\_free  
  init.c, [211](#)  
fasp\_ilu\_data\_null  
  init.c, [211](#)  
fasp\_ilu\_dbsr\_setup  
  ilu\_setup\_bsr.c, [205](#)  
fasp\_ilu\_dcsr\_setup  
  ilu\_setup\_csr.c, [206](#)  
fasp\_ilu\_dstr\_setup0  
  ilu\_setup\_str.c, [207](#)  
fasp\_ilu\_dstr\_setup1  
  ilu\_setup\_str.c, [208](#)  
fasp\_ivec\_alloc  
  vec.c, [462](#)  
fasp\_ivec\_create  
  vec.c, [463](#)  
fasp\_ivec\_free  
  vec.c, [463](#)  
fasp\_ivec\_print  
  io.c, [238](#)  
fasp\_ivec\_read  
  io.c, [238](#)  
fasp\_ivec\_set  
  vec.c, [463](#)  
fasp\_ivec\_write  
  io.c, [239](#)  
fasp\_ivecind\_read  
  io.c, [239](#)  
fasp\_matrix\_read  
  io.c, [240](#)  
fasp\_matrix\_read\_bin  
  io.c, [240](#)  
fasp\_matrix\_write  
  io.c, [241](#)  
fasp\_mem\_calloc  
  memory.c, [267](#)  
fasp\_mem\_check  
  memory.c, [267](#)  
fasp\_mem\_dcsr\_check  
  memory.c, [267](#)  
fasp\_mem\_free  
  memory.c, [268](#)  
fasp\_mem\_iludata\_check  
  memory.c, [268](#)  
fasp\_mem\_realloc  
  memory.c, [269](#)  
fasp\_mem\_usage  
  memory.c, [269](#)  
fasp\_param\_Schwarz\_init  
  parameters.c, [286](#)  
fasp\_param\_Schwarz\_print  
  parameters.c, [288](#)  
fasp\_param\_Schwarz\_set  
  parameters.c, [288](#)  
fasp\_param\_amg\_init

parameters.c, [282](#)  
 fasp\_param\_amg\_print  
   parameters.c, [283](#)  
 fasp\_param\_amg\_set  
   parameters.c, [283](#)  
 fasp\_param\_amg\_to\_prec  
   parameters.c, [283](#)  
 fasp\_param\_amg\_to\_prec\_bsr  
   parameters.c, [284](#)  
 fasp\_param\_check  
   input.c, [213](#)  
 fasp\_param\_ilu\_init  
   parameters.c, [284](#)  
 fasp\_param\_ilu\_print  
   parameters.c, [284](#)  
 fasp\_param\_ilu\_set  
   parameters.c, [285](#)  
 fasp\_param\_init  
   parameters.c, [285](#)  
 fasp\_param\_input  
   input.c, [214](#)  
 fasp\_param\_input\_init  
   parameters.c, [285](#)  
 fasp\_param\_prec\_to\_amg  
   parameters.c, [286](#)  
 fasp\_param\_prec\_to\_amg\_bsr  
   parameters.c, [286](#)  
 fasp\_param\_set  
   parameters.c, [288](#)  
 fasp\_param\_solver\_init  
   parameters.c, [289](#)  
 fasp\_param\_solver\_print  
   parameters.c, [289](#)  
 fasp\_param\_solver\_set  
   parameters.c, [289](#)  
 fasp\_poisson\_fgmg\_1D  
   gmg\_poisson.c, [196](#)  
 fasp\_poisson\_fgmg\_2D  
   gmg\_poisson.c, [196](#)  
 fasp\_poisson\_fgmg\_3D  
   gmg\_poisson.c, [197](#)  
 fasp\_poisson\_gmg\_1D  
   gmg\_poisson.c, [197](#)  
 fasp\_poisson\_gmg\_2D  
   gmg\_poisson.c, [198](#)  
 fasp\_poisson\_gmg\_3D  
   gmg\_poisson.c, [198](#)  
 fasp\_poisson\_pcg\_gmg\_1D  
   gmg\_poisson.c, [199](#)  
 fasp\_poisson\_pcg\_gmg\_2D  
   gmg\_poisson.c, [199](#)  
 fasp\_poisson\_pcg\_gmg\_3D  
   gmg\_poisson.c, [201](#)  
 fasp\_precond\_Schwarz  
   precond\_csr.c, [339](#)  
 fasp\_precond\_amg  
   precond\_csr.c, [335](#)  
 fasp\_precond\_amg\_nk  
   precond\_csr.c, [336](#)  
 fasp\_precond\_amli  
   precond\_csr.c, [336](#)  
 fasp\_precond\_block\_SGS\_3  
   precond\_bcsr.c, [323](#)  
 fasp\_precond\_block\_SGS\_3\_amg  
   precond\_bcsr.c, [325](#)  
 fasp\_precond\_block\_diag\_3  
   precond\_bcsr.c, [320](#)  
 fasp\_precond\_block\_diag\_3\_amg  
   precond\_bcsr.c, [321](#)  
 fasp\_precond\_block\_diag\_4  
   precond\_bcsr.c, [321](#)  
 fasp\_precond\_block\_lower\_3  
   precond\_bcsr.c, [321](#)  
 fasp\_precond\_block\_lower\_3\_amg  
   precond\_bcsr.c, [323](#)  
 fasp\_precond\_block\_lower\_4  
   precond\_bcsr.c, [323](#)  
 fasp\_precond\_block\_upper\_3  
   precond\_bcsr.c, [325](#)  
 fasp\_precond\_block\_upper\_3\_amg  
   precond\_bcsr.c, [325](#)  
 fasp\_precond\_data\_null  
   init.c, [212](#)  
 fasp\_precond\_dbsr\_amg  
   precond\_bsr.c, [328](#)  
 fasp\_precond\_dbsr\_amg\_nk  
   precond\_bsr.c, [328](#)  
 fasp\_precond\_dbsr\_diag  
   precond\_bsr.c, [329](#)  
 fasp\_precond\_dbsr\_diag\_nc2  
   precond\_bsr.c, [329](#)  
 fasp\_precond\_dbsr\_diag\_nc3  
   precond\_bsr.c, [330](#)  
 fasp\_precond\_dbsr\_diag\_nc5  
   precond\_bsr.c, [330](#)  
 fasp\_precond\_dbsr\_diag\_nc7  
   precond\_bsr.c, [332](#)  
 fasp\_precond\_dbsr\_ilu  
   precond\_bsr.c, [332](#)  
 fasp\_precond\_dbsr\_nl\_amli  
   precond\_bsr.c, [334](#)  
 fasp\_precond\_diag  
   precond\_csr.c, [336](#)  
 fasp\_precond\_dstr\_blockgs  
   precond\_str.c, [341](#)  
 fasp\_precond\_dstr\_diag  
   precond\_str.c, [341](#)  
 fasp\_precond\_dstr\_ilu0

precond\_str.c, [341](#)  
fasp\_precond\_dstr\_ilu0\_backward  
precond\_str.c, [342](#)  
fasp\_precond\_dstr\_ilu0\_forward  
precond\_str.c, [342](#)  
fasp\_precond\_dstr\_ilu1  
precond\_str.c, [342](#)  
fasp\_precond\_dstr\_ilu1\_backward  
precond\_str.c, [344](#)  
fasp\_precond\_dstr\_ilu1\_forward  
precond\_str.c, [344](#)  
fasp\_precond\_famg  
precond\_csr.c, [337](#)  
fasp\_precond\_free  
precond\_csr.c, [337](#)  
fasp\_precond\_ilu  
precond\_csr.c, [337](#)  
fasp\_precond\_ilu\_backward  
precond\_csr.c, [338](#)  
fasp\_precond\_ilu\_forward  
precond\_csr.c, [338](#)  
fasp\_precond\_nl\_amli  
precond\_csr.c, [339](#)  
fasp\_precond\_null  
init.c, [212](#)  
fasp\_precond\_setup  
precond\_csr.c, [339](#)  
fasp\_precond\_sweeping  
precond\_bcsr.c, [327](#)  
fasp\_quad2d  
quadrature.c, [355](#)  
fasp\_set\_GS\_threads  
threads.c, [454](#)  
fasp\_smat\_identity  
smat.c, [363](#)  
fasp\_smat\_identity\_nc2  
smat.c, [363](#)  
fasp\_smat\_identity\_nc3  
smat.c, [365](#)  
fasp\_smat\_identity\_nc5  
smat.c, [365](#)  
fasp\_smat\_identity\_nc7  
smat.c, [365](#)  
fasp\_smat\_lu\_decomp  
lu.c, [264](#)  
fasp\_smat\_lu\_solve  
lu.c, [265](#)  
fasp\_smoother\_dbsr\_gs  
smoother\_bsr.c, [367](#)  
fasp\_smoother\_dbsr\_gs1  
smoother\_bsr.c, [367](#)  
fasp\_smoother\_dbsr\_gs\_ascend  
smoother\_bsr.c, [368](#)  
fasp\_smoother\_dbsr\_gs\_ascend1  
smoother\_bsr.c, [368](#)  
fasp\_smoother\_dbsr\_gs\_descend  
smoother\_bsr.c, [369](#)  
fasp\_smoother\_dbsr\_gs\_descend1  
smoother\_bsr.c, [369](#)  
fasp\_smoother\_dbsr\_gs\_order1  
smoother\_bsr.c, [370](#)  
fasp\_smoother\_dbsr\_gs\_order2  
smoother\_bsr.c, [370](#)  
fasp\_smoother\_dbsr\_ilu  
smoother\_bsr.c, [371](#)  
fasp\_smoother\_dbsr\_jacobi  
smoother\_bsr.c, [371](#)  
fasp\_smoother\_dbsr\_jacobi1  
smoother\_bsr.c, [371](#)  
fasp\_smoother\_dbsr\_jacobi\_setup  
smoother\_bsr.c, [373](#)  
fasp\_smoother\_dbsr\_sor  
smoother\_bsr.c, [373](#)  
fasp\_smoother\_dbsr\_sor1  
smoother\_bsr.c, [374](#)  
fasp\_smoother\_dbsr\_sor\_ascend  
smoother\_bsr.c, [374](#)  
fasp\_smoother\_dbsr\_sor\_descend  
smoother\_bsr.c, [375](#)  
fasp\_smoother\_dbsr\_sor\_order  
smoother\_bsr.c, [375](#)  
fasp\_smoother\_dcsr\_L1diag  
smoother\_csr.c, [380](#)  
fasp\_smoother\_dcsr\_gs  
smoother\_csr.c, [377](#)  
fasp\_smoother\_dcsr\_gs\_cf  
smoother\_csr.c, [377](#)  
fasp\_smoother\_dcsr\_gs\_rb3d  
smoother\_csr.c, [378](#)  
fasp\_smoother\_dcsr\_gscr  
smoother\_csr\_cr.c, [382](#)  
fasp\_smoother\_dcsr\_ilu  
smoother\_csr.c, [378](#)  
fasp\_smoother\_dcsr\_jacobi  
smoother\_csr.c, [379](#)  
fasp\_smoother\_dcsr\_kaczmarz  
smoother\_csr.c, [379](#)  
fasp\_smoother\_dcsr\_poly  
smoother\_csr\_poly.c, [384](#)  
fasp\_smoother\_dcsr\_poly\_old  
smoother\_csr\_poly.c, [384](#)  
fasp\_smoother\_dcsr\_sgs  
smoother\_csr.c, [380](#)  
fasp\_smoother\_dcsr\_sor  
smoother\_csr.c, [381](#)  
fasp\_smoother\_dcsr\_sor\_cf  
smoother\_csr.c, [381](#)  
fasp\_smoother\_dstr\_gs

smoother\_str.c, [386](#)  
 fasp\_smoother\_dstr\_gs1  
   smoother\_str.c, [386](#)  
 fasp\_smoother\_dstr\_gs\_ascend  
   smoother\_str.c, [387](#)  
 fasp\_smoother\_dstr\_gs\_cf  
   smoother\_str.c, [387](#)  
 fasp\_smoother\_dstr\_gs\_descend  
   smoother\_str.c, [388](#)  
 fasp\_smoother\_dstr\_gs\_order  
   smoother\_str.c, [388](#)  
 fasp\_smoother\_dstr\_jacobi  
   smoother\_str.c, [389](#)  
 fasp\_smoother\_dstr\_jacobi1  
   smoother\_str.c, [389](#)  
 fasp\_smoother\_dstr\_schwarz  
   smoother\_str.c, [390](#)  
 fasp\_smoother\_dstr\_sor  
   smoother\_str.c, [390](#)  
 fasp\_smoother\_dstr\_sor1  
   smoother\_str.c, [390](#)  
 fasp\_smoother\_dstr\_sor\_ascend  
   smoother\_str.c, [391](#)  
 fasp\_smoother\_dstr\_sor\_cf  
   smoother\_str.c, [391](#)  
 fasp\_smoother\_dstr\_sor\_descend  
   smoother\_str.c, [392](#)  
 fasp\_smoother\_dstr\_sor\_order  
   smoother\_str.c, [392](#)  
 fasp\_solver\_amg  
   amg.c, [77](#)  
 fasp\_solver\_amli  
   amlirecur.c, [88](#)  
 fasp\_solver\_bdcscr\_itsolver  
   itsolver\_bcsr.c, [244](#)  
 fasp\_solver\_bdcscr\_krylov  
   itsolver\_bcsr.c, [244](#)  
 fasp\_solver\_bdcscr\_krylov\_block\_3  
   itsolver\_bcsr.c, [245](#)  
 fasp\_solver\_bdcscr\_krylov\_block\_4  
   itsolver\_bcsr.c, [245](#)  
 fasp\_solver\_bdcscr\_krylov\_sweeping  
   itsolver\_bcsr.c, [246](#)  
 fasp\_solver\_bdcscr\_pbcgs  
   pbcgs.c, [291](#)  
 fasp\_solver\_bdcscr\_pcg  
   pcg.c, [298](#)  
 fasp\_solver\_bdcscr\_pgmres  
   pgmres.c, [309](#)  
 fasp\_solver\_bdcscr\_pminres  
   pminres.c, [315](#)  
 fasp\_solver\_bdcscr\_pvfgmres  
   pvfgmres.c, [345](#)  
 fasp\_solver\_bdcscr\_pvgmres  
   pvgmres.c, [349](#)  
 fasp\_solver\_bdcscr\_spgmres  
   spbcgs.c, [435](#)  
 fasp\_solver\_bdcscr\_spcg  
   spcg.c, [440](#)  
 fasp\_solver\_bdcscr\_spgmres  
   spgmres.c, [442](#)  
 fasp\_solver\_bdcscr\_spmminres  
   spminres.c, [447](#)  
 fasp\_solver\_bdcscr\_spvgmres  
   spvgmres.c, [451](#)  
 fasp\_solver\_dbsr\_itsolver  
   itsolver\_bsr.c, [247](#)  
 fasp\_solver\_dbsr\_krylov  
   itsolver\_bsr.c, [248](#)  
 fasp\_solver\_dbsr\_krylov\_amg  
   itsolver\_bsr.c, [248](#)  
 fasp\_solver\_dbsr\_krylov\_amg\_nk  
   itsolver\_bsr.c, [249](#)  
 fasp\_solver\_dbsr\_krylov\_diag  
   itsolver\_bsr.c, [249](#)  
 fasp\_solver\_dbsr\_krylov\_ilu  
   itsolver\_bsr.c, [250](#)  
 fasp\_solver\_dbsr\_krylov\_nk\_amg  
   itsolver\_bsr.c, [250](#)  
 fasp\_solver\_dbsr\_pbcgs  
   pbcgs.c, [292](#)  
 fasp\_solver\_dbsr\_pcg  
   pcg.c, [299](#)  
 fasp\_solver\_dbsr\_pgmres  
   pgmres.c, [310](#)  
 fasp\_solver\_dbsr\_pvfgmres  
   pvfgmres.c, [346](#)  
 fasp\_solver\_dbsr\_pvgmres  
   pvgmres.c, [350](#)  
 fasp\_solver\_dbsr\_spgmres  
   spbcgs.c, [435](#)  
 fasp\_solver\_dbsr\_spgmres  
   spgmres.c, [443](#)  
 fasp\_solver\_dbsr\_spvgmres  
   spvgmres.c, [451](#)  
 fasp\_solver\_dcsr\_itsolver  
   itsolver\_csr.c, [252](#)  
 fasp\_solver\_dcsr\_krylov  
   itsolver\_csr.c, [252](#)  
 fasp\_solver\_dcsr\_krylov\_Schwarz  
   itsolver\_csr.c, [255](#)  
 fasp\_solver\_dcsr\_krylov\_amg  
   itsolver\_csr.c, [253](#)  
 fasp\_solver\_dcsr\_krylov\_amg\_nk  
   itsolver\_csr.c, [253](#)  
 fasp\_solver\_dcsr\_krylov\_diag  
   itsolver\_csr.c, [254](#)  
 fasp\_solver\_dcsr\_krylov\_ilu

itsolver\_csr.c, [254](#)  
fasp\_solver\_dcsr\_krylov\_ilu\_M  
itsolver\_csr.c, [255](#)  
fasp\_solver\_dcsr\_pbcgs  
pbcgs.c, [293](#)  
fasp\_solver\_dcsr\_pcg  
pcg.c, [299](#)  
fasp\_solver\_dcsr\_pgcg  
pgcg.c, [304](#)  
fasp\_solver\_dcsr\_pgcr  
pgcr.c, [307](#)  
fasp\_solver\_dcsr\_pgcr1  
pgcr.c, [308](#)  
fasp\_solver\_dcsr\_pgmres  
pgmres.c, [311](#)  
fasp\_solver\_dcsr\_pminres  
pminres.c, [315](#)  
fasp\_solver\_dcsr\_pvfgmres  
pvfgmres.c, [347](#)  
fasp\_solver\_dcsr\_pvgmres  
pvgmres.c, [350](#)  
fasp\_solver\_dcsr\_spbcgs  
spbcgs.c, [436](#)  
fasp\_solver\_dcsr\_spcg  
spcg.c, [440](#)  
fasp\_solver\_dcsr\_spgmres  
spgmres.c, [443](#)  
fasp\_solver\_dcsr\_spmminres  
spmminres.c, [448](#)  
fasp\_solver\_dcsr\_spvgmres  
spvgmres.c, [452](#)  
fasp\_solver\_dstr\_itsolver  
itsolver\_str.c, [261](#)  
fasp\_solver\_dstr\_krylov  
itsolver\_str.c, [261](#)  
fasp\_solver\_dstr\_krylov\_blockgs  
itsolver\_str.c, [261](#)  
fasp\_solver\_dstr\_krylov\_diag  
itsolver\_str.c, [263](#)  
fasp\_solver\_dstr\_krylov\_ilu  
itsolver\_str.c, [263](#)  
fasp\_solver\_dstr\_pbcgs  
pbcgs.c, [293](#)  
fasp\_solver\_dstr\_pcg  
pcg.c, [301](#)  
fasp\_solver\_dstr\_pgmres  
pgmres.c, [311](#)  
fasp\_solver\_dstr\_pminres  
pminres.c, [316](#)  
fasp\_solver\_dstr\_pvgmres  
pvgmres.c, [352](#)  
fasp\_solver\_dstr\_spbcgs  
spbcgs.c, [436](#)  
fasp\_solver\_dstr\_spcg  
spcg.c, [441](#)  
fasp\_solver\_dstr\_spgmres  
spgmres.c, [445](#)  
fasp\_solver\_dstr\_spmminres  
spmminres.c, [448](#)  
fasp\_solver\_dstr\_spvgmres  
spvgmres.c, [452](#)  
fasp\_solver\_famg  
famg.c, [157](#)  
fasp\_solver\_fmecycle  
fmecycle.c, [189](#)  
fasp\_solver\_itsolver  
itsolver\_mf.c, [258](#)  
fasp\_solver\_itsolver\_init  
itsolver\_mf.c, [259](#)  
fasp\_solver\_krylov  
itsolver\_mf.c, [259](#)  
fasp\_solver\_mgcycle  
mgcycle.c, [273](#)  
fasp\_solver\_mgcycle\_bsr  
mgcycle.c, [273](#)  
fasp\_solver\_mgrecur  
mgrecur.c, [274](#)  
fasp\_solver\_mumps  
interface\_mumps.c, [215](#)  
fasp\_solver\_mumps\_steps  
interface\_mumps.c, [215](#)  
fasp\_solver\_nl\_amli  
amlirecur.c, [89](#)  
fasp\_solver\_nl\_amli\_bsr  
amlirecur.c, [89](#)  
fasp\_solver\_pbcgs  
pbcgs\_mf.c, [295](#)  
fasp\_solver\_pcg  
pcg\_mf.c, [303](#)  
fasp\_solver\_pgcg  
pgcg\_mf.c, [305](#)  
fasp\_solver\_pgmres  
pgmres\_mf.c, [313](#)  
fasp\_solver\_pminres  
pminres\_mf.c, [318](#)  
fasp\_solver\_pvfgmres  
pvfgmres\_mf.c, [348](#)  
fasp\_solver\_pvgmres  
pvgmres\_mf.c, [353](#)  
fasp\_solver\_superlu  
interface\_superlu.c, [217](#)  
fasp\_solver\_umfpack  
interface\_umfpack.c, [218](#)  
fasp\_sparse\_MIS  
sparse\_util.c, [430](#)  
fasp\_sparse\_aat  
sparse\_util.c, [427](#)  
fasp\_sparse\_abyb\_

- sparse\_util.c, [428](#)
- fasp\_sparse\_abybms\_
  - sparse\_util.c, [428](#)
- fasp\_sparse\_aplbms\_
  - sparse\_util.c, [429](#)
- fasp\_sparse\_aplusb\_
  - sparse\_util.c, [429](#)
- fasp\_sparse\_iit\_
  - sparse\_util.c, [429](#)
- fasp\_sparse\_rapcmp\_
  - sparse\_util.c, [430](#)
- fasp\_sparse\_rapms\_
  - sparse\_util.c, [431](#)
- fasp\_sparse\_wta\_
  - sparse\_util.c, [431](#)
- fasp\_sparse\_wtams\_
  - sparse\_util.c, [432](#)
- fasp\_sparse\_ytx\_
  - sparse\_util.c, [432](#)
- fasp\_sparse\_ytxbig\_
  - sparse\_util.c, [433](#)
- fasp\_vector\_read
  - io.c, [241](#)
- fasp\_vector\_write
  - io.c, [242](#)
- fasp\_wrapper\_dbsr\_krylov\_amg
  - wrapper.c, [466](#)
- fasp\_wrapper\_dcoo\_dbsr\_krylov\_amg
  - wrapper.c, [466](#)
- fmgcycle.c, [188](#)
  - fasp\_solver\_fmgcycle, [189](#)
- formats.c, [189](#)
  - fasp\_format\_bdcsr\_dcsr, [190](#)
  - fasp\_format\_dbsr\_dcoo, [190](#)
  - fasp\_format\_dbsr\_dcsr, [191](#)
  - fasp\_format\_dcoo\_dcsr, [191](#)
  - fasp\_format\_dcsr\_dbsr, [192](#)
  - fasp\_format\_dcsr\_dcoo, [192](#)
  - fasp\_format\_dcsr\_dcsr, [193](#)
  - fasp\_format\_dstr\_dbsr, [193](#)
  - fasp\_format\_dstr\_dcsr, [194](#)
- G0PT
  - fasp\_const.h, [177](#)
- GE
  - fasp.h, [162](#)
- GT
  - fasp.h, [162](#)
- givens.c, [194](#)
  - fasp\_aux\_givens, [195](#)
- gmg\_poisson.c, [195](#)
  - fasp\_poisson\_fgmg\_1D, [196](#)
  - fasp\_poisson\_fgmg\_2D, [196](#)
  - fasp\_poisson\_fgmg\_3D, [197](#)

- fasp\_poisson\_gmg\_1D, [197](#)
  - fasp\_poisson\_gmg\_2D, [198](#)
  - fasp\_poisson\_gmg\_3D, [198](#)
  - fasp\_poisson\_pcg\_gmg\_1D, [199](#)
  - fasp\_poisson\_pcg\_gmg\_2D, [199](#)
  - fasp\_poisson\_pcg\_gmg\_3D, [201](#)
- graphics.c, [202](#)
  - fasp\_dbsr\_plot, [202](#)
  - fasp\_dbsr\_subplot, [203](#)
  - fasp\_dcsr\_plot, [203](#)
  - fasp\_dcsr\_subplot, [204](#)
  - fasp\_grid2d\_plot, [204](#)
- grid2d, [36](#)
  - e, [37](#)
  - edges, [37](#)
  - ediri, [37](#)
  - efather, [37](#)
  - fasp.h, [164](#)
  - p, [37](#)
  - pdiri, [37](#)
  - pfather, [38](#)
  - s, [38](#)
  - t, [38](#)
  - tfather, [38](#)
  - triangles, [38](#)
  - vertices, [38](#)
- ICNTL
  - interface\_mumps.c, [215](#)
- iCOOmat, [38](#)
  - fasp.h, [164](#)
- iCSRmat, [39](#)
  - fasp.h, [164](#)
- ILU\_data, [40](#)
- ILU\_droptol
  - input\_param, [47](#)
- ILU\_lfil
  - input\_param, [47](#)
- ILU\_param, [41](#)
- ILU\_permtol
  - input\_param, [47](#)
- ILU\_relax
  - input\_param, [47](#)
- ILU\_type
  - input\_param, [47](#)
- ILUk
  - fasp\_const.h, [178](#)
- ILUt
  - fasp\_const.h, [178](#)
- ILUtp
  - fasp\_const.h, [178](#)
- IMAP
  - fasp.h, [165](#)
- INT

- fasp.h, 162
- INTERP\_DIR
  - fasp\_const.h, 178
- INTERP\_ENG
  - fasp\_const.h, 178
- INTERP\_STD
  - fasp\_const.h, 178
- ISNAN
  - fasp.h, 162
- ISPT
  - fasp\_const.h, 178
- idenmat, 40
  - fasp.h, 165
- ilength
  - io.c, 243
- ilu\_setup\_bsr.c, 205
  - fasp\_ilu\_dbsr\_setup, 205
- ilu\_setup\_csr.c, 206
  - fasp\_ilu\_dcsr\_setup, 206
- ilu\_setup\_str.c, 207
  - fasp\_ilu\_dstr\_setup0, 207
  - fasp\_ilu\_dstr\_setup1, 208
- inifile
  - input\_param, 47
- init.c, 208
  - fasp\_Schwarz\_data\_free, 212
  - fasp\_amg\_data\_bsr\_create, 209
  - fasp\_amg\_data\_bsr\_free, 209
  - fasp\_amg\_data\_create, 210
  - fasp\_amg\_data\_free, 210
  - fasp\_ilu\_data\_alloc, 211
  - fasp\_ilu\_data\_free, 211
  - fasp\_ilu\_data\_null, 211
  - fasp\_precond\_data\_null, 212
  - fasp\_precond\_null, 212
- input.c, 213
  - fasp\_param\_check, 213
  - fasp\_param\_input, 214
- input\_param, 42
  - AMG\_ILU\_levels, 44
  - AMG\_Schwarz\_levels, 45
  - AMG\_aggregation\_type, 43
  - AMG\_aggressive\_level, 43
  - AMG\_aggressive\_path, 43
  - AMG\_amli\_degree, 43
  - AMG\_coarse\_dof, 43
  - AMG\_coarse\_scaling, 44
  - AMG\_coarse\_solver, 44
  - AMG\_coarsening\_type, 44
  - AMG\_cycle\_type, 44
  - 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, 45
  - AMG\_quality\_bound, 45
  - AMG\_relaxation, 45
  - AMG\_smooth\_filter, 46
  - AMG\_smooth\_order, 46
  - AMG\_smoother, 46
  - AMG\_strong\_coupled, 46
  - AMG\_strong\_threshold, 46
  - AMG\_tentative\_smooth, 46
  - AMG\_tol, 46
  - AMG\_truncation\_threshold, 46
  - AMG\_type, 46
  - ILU\_droptol, 47
  - ILU\_lfil, 47
  - ILU\_permtol, 47
  - ILU\_relax, 47
  - ILU\_type, 47
  - inifile, 47
  - itsolver\_maxit, 47
  - itsolver\_tol, 47
  - output\_type, 47
  - precond\_type, 48
  - print\_level, 48
  - problem\_num, 48
  - restart, 48
  - Schwarz\_blk solver, 48
  - Schwarz\_maxlvl, 48
  - Schwarz\_mmsize, 48
  - Schwarz\_type, 48
  - solver\_type, 48
  - stop\_type, 49
  - workdir, 49
- interface\_mumps.c, 214
  - fasp\_solver\_mumps, 215
  - fasp\_solver\_mumps\_steps, 215
  - ICNTL, 215
- interface\_samg.c, 216
  - dCSRmat2SAMGInput, 216
  - dvector2SAMGInput, 217
- interface\_superlu.c, 217
  - fasp\_solver\_superlu, 217
- interface\_umfpack.c, 218
  - fasp\_solver\_umfpack, 218
- interpolation.c, 219
  - fasp\_amg\_interp, 219
  - fasp\_amg\_interp1, 220
  - fasp\_amg\_interp\_trunc, 220
- interpolation\_em.c, 221
  - fasp\_amg\_interp\_em, 221

- io.c, 222
  - dlength, 243
  - fasp\_dbsr\_print, 224
  - fasp\_dbsr\_read, 224
  - fasp\_dbsr\_write, 225
  - fasp\_dbsr\_write\_coo, 225
  - fasp\_dcoo1\_read, 226
  - fasp\_dcoo\_print, 226
  - fasp\_dcoo\_read, 227
  - fasp\_dcoo\_shift\_read, 227
  - fasp\_dcoo\_write, 228
  - fasp\_dcsr\_print, 228
  - fasp\_dcsr\_read, 229
  - fasp\_dcsr\_write\_coo, 229
  - fasp\_dcsrvec1\_read, 229
  - fasp\_dcsrvec1\_write, 230
  - fasp\_dcsrvec2\_read, 231
  - fasp\_dcsrvec2\_write, 231
  - fasp\_dmtx\_read, 232
  - fasp\_dmtxsym\_read, 232
  - fasp\_dstr\_print, 234
  - fasp\_dstr\_read, 234
  - fasp\_dstr\_write, 235
  - fasp\_dvec\_print, 235
  - fasp\_dvec\_read, 236
  - fasp\_dvec\_write, 236
  - fasp\_dvecind\_read, 236
  - fasp\_dvecind\_write, 237
  - fasp\_hb\_read, 237
  - fasp\_ivec\_print, 238
  - fasp\_ivec\_read, 238
  - fasp\_ivec\_write, 239
  - fasp\_ivecind\_read, 239
  - fasp\_matrix\_read, 240
  - fasp\_matrix\_read\_bin, 240
  - fasp\_matrix\_write, 241
  - fasp\_vector\_read, 241
  - fasp\_vector\_write, 242
  - ilength, 243
- itsolver\_bcsr.c, 243
  - fasp\_solver\_bdcsr\_itsolver, 244
  - fasp\_solver\_bdcsr\_krylov, 244
  - fasp\_solver\_bdcsr\_krylov\_block\_3, 245
  - fasp\_solver\_bdcsr\_krylov\_block\_4, 245
  - fasp\_solver\_bdcsr\_krylov\_sweeping, 246
- itsolver\_bsr.c, 247
  - fasp\_solver\_dbsr\_itsolver, 247
  - fasp\_solver\_dbsr\_krylov, 248
  - fasp\_solver\_dbsr\_krylov\_amg, 248
  - fasp\_solver\_dbsr\_krylov\_amg\_nk, 249
  - fasp\_solver\_dbsr\_krylov\_diag, 249
  - fasp\_solver\_dbsr\_krylov\_ilu, 250
  - fasp\_solver\_dbsr\_krylov\_nk\_amg, 250
- itsolver\_csr.c, 251
  - fasp\_solver\_dcsr\_itsolver, 252
  - fasp\_solver\_dcsr\_krylov, 252
  - fasp\_solver\_dcsr\_krylov\_Schwarz, 255
  - fasp\_solver\_dcsr\_krylov\_amg, 253
  - fasp\_solver\_dcsr\_krylov\_amg\_nk, 253
  - fasp\_solver\_dcsr\_krylov\_diag, 254
  - fasp\_solver\_dcsr\_krylov\_ilu, 254
  - fasp\_solver\_dcsr\_krylov\_ilu\_M, 255
- itsolver\_maxit
  - input\_param, 47
- itsolver\_mf.c, 257
  - fasp\_solver\_itsolver, 258
  - fasp\_solver\_itsolver\_init, 259
  - fasp\_solver\_krylov, 259
- itsolver\_param, 49
  - itsolver\_type, 49
  - maxit, 49
  - precond\_type, 50
  - print\_level, 50
  - restart, 50
  - stop\_type, 50
  - tol, 50
- itsolver\_str.c, 260
  - fasp\_solver\_dstr\_itsolver, 261
  - fasp\_solver\_dstr\_krylov, 261
  - fasp\_solver\_dstr\_krylov\_blockgs, 261
  - fasp\_solver\_dstr\_krylov\_diag, 263
  - fasp\_solver\_dstr\_krylov\_ilu, 263
- itsolver\_tol
  - input\_param, 47
- itsolver\_type
  - itsolver\_param, 49
- ivector, 50
  - fasp.h, 165
- JA
  - dBSRmat, 32
- LE
  - fasp.h, 162
- LONG
  - fasp.h, 162
- LONGLONG
  - fasp.h, 162
- LS
  - fasp.h, 162
- LU\_diag
  - precond\_block\_data, 57
- Link, 51
- LinkedList
  - fasp.h, 165
- linked\_list, 51
- ListElement
  - fasp.h, 165
- local\_A



- precond\_sweeping\_data, 72
- local\_LU
  - precond\_sweeping\_data, 72
- local\_index
  - precond\_sweeping\_data, 72
- lu.c, 264
  - fasp\_smat\_lu\_decomp, 264
  - fasp\_smat\_lu\_solve, 265
- MAT\_BSR
  - fasp\_const.h, 179
- MAT\_CSR
  - fasp\_const.h, 179
- MAT\_CSRL
  - fasp\_const.h, 179
- MAT\_FREE
  - fasp\_const.h, 179
- MAT\_STR
  - fasp\_const.h, 179
- MAT\_SymCSR
  - fasp\_const.h, 179
- MAT\_bBSR
  - fasp\_const.h, 178
- MAT\_bCSR
  - fasp\_const.h, 179
- MAX
  - fasp.h, 163
- MAX\_AMG\_LVL
  - fasp\_const.h, 179
- MAX\_CRATE
  - fasp\_const.h, 180
- MAX\_REFINE\_LVL
  - fasp\_const.h, 180
- MAX\_RESTART
  - fasp\_const.h, 180
- MAX\_STAG
  - fasp\_const.h, 180
- MAXIMAP
  - fasp.h, 165
- MIN
  - fasp.h, 163
- MIN\_CDOF
  - fasp\_const.h, 180
- MIN\_CRATE
  - fasp\_const.h, 180
- mallinfo, 52
- malloc\_chunk, 52
- malloc\_params, 53
- malloc\_segment, 53
- malloc\_state, 53
- malloc\_tree\_chunk, 54
- maxit
  - itsolver\_param, 49
  - precond\_FASP\_blkoi\_data, 69
- memory.c, 266
  - fasp\_mem\_calloc, 267
  - fasp\_mem\_check, 267
  - fasp\_mem\_dcsr\_check, 267
  - fasp\_mem\_free, 268
  - fasp\_mem\_iludata\_check, 268
  - fasp\_mem\_realloc, 269
  - fasp\_mem\_usage, 269
  - total\_alloc\_count, 269
  - total\_alloc\_mem, 269
- message.c, 270
  - fasp\_chkerr, 270
  - print\_amgcomplexity, 271
  - print\_amgcomplexity\_bsr, 271
  - print\_cputime, 271
  - print\_itinfo, 272
  - print\_message, 272
- mgcycle.c, 273
  - fasp\_solver\_mgcycle, 273
  - fasp\_solver\_mgcycle\_bsr, 273
- mgl
  - precond\_block\_data, 57
- mgl\_data
  - precond\_FASP\_blkoi\_data, 69
- mgrecur.c, 274
  - fasp\_solver\_mgrecur, 274
- Mumps\_data, 55
- mxv\_matfree, 55
- NEDMALLOC
  - fasp.h, 163
- NL\_AMLI\_CYCLE
  - fasp\_const.h, 180
- NO\_ORDER
  - fasp\_const.h, 180
- nedmallinfo, 55
- neigh
  - precond\_FASP\_blkoi\_data, 69
- NumLayers
  - precond\_sweeping\_data, 72
- nx\_rb
  - fasp.h, 165
- ny\_rb
  - fasp.h, 166
- nz\_rb
  - fasp.h, 166
- OFF
  - fasp\_const.h, 181
- ON
  - fasp\_const.h, 181
- OPENMP\_HOLDS
  - fasp\_const.h, 181
- order
  - precond\_FASP\_blkoi\_data, 69

- precond\_block\_reservoir\_data, 60
- ordering.c, 275
  - fasp\_BinarySearch, 280
  - fasp\_aux\_dQuickSort, 276
  - fasp\_aux\_dQuickSortIndex, 276
  - fasp\_aux\_iQuickSort, 276
  - fasp\_aux\_iQuickSortIndex, 278
  - fasp\_aux\_merge, 278
  - fasp\_aux\_msort, 279
  - fasp\_aux\_unique, 279
  - fasp\_dcsr\_CMK\_order, 280
  - fasp\_dcsr\_RCMK\_order, 281
- output\_type
  - input\_param, 47
- p
  - grid2d, 37
- PAIRWISE
  - fasp\_const.h, 181
- PP
  - precond\_FASP\_blkoi\_data, 70
  - precond\_block\_reservoir\_data, 60
- PREC\_AMG
  - fasp\_const.h, 181
- PREC\_DIAG
  - fasp\_const.h, 181
- PREC\_FMG
  - fasp\_const.h, 181
- PREC\_ILU
  - fasp\_const.h, 181
- PREC\_NULL
  - fasp\_const.h, 182
- PREC\_SCHWARZ
  - fasp\_const.h, 182
- PRINT\_ALL
  - fasp\_const.h, 182
- PRINT\_MIN
  - fasp\_const.h, 182
- PRINT\_MORE
  - fasp\_const.h, 182
- PRINT\_MOST
  - fasp\_const.h, 182
- PRINT\_NONE
  - fasp\_const.h, 182
- PRINT\_SOME
  - fasp\_const.h, 182
- PUT\_INT
  - fasp.h, 163
- PUT\_REAL
  - fasp.h, 163
- parameters.c, 281
  - fasp\_param\_Schwarz\_init, 286
  - fasp\_param\_Schwarz\_print, 288
  - fasp\_param\_Schwarz\_set, 288
  - fasp\_param\_amg\_init, 282
  - fasp\_param\_amg\_print, 283
  - fasp\_param\_amg\_set, 283
  - fasp\_param\_amg\_to\_prec, 283
  - fasp\_param\_amg\_to\_prec\_bsr, 284
  - fasp\_param\_ilu\_init, 284
  - fasp\_param\_ilu\_print, 284
  - fasp\_param\_ilu\_set, 285
  - fasp\_param\_init, 285
  - fasp\_param\_input\_init, 285
  - fasp\_param\_prec\_to\_amg, 286
  - fasp\_param\_prec\_to\_amg\_bsr, 286
  - fasp\_param\_set, 288
  - fasp\_param\_solver\_init, 289
  - fasp\_param\_solver\_print, 289
  - fasp\_param\_solver\_set, 289
- pbcgs.c, 290
  - fasp\_solver\_bdcsr\_pbcgs, 291
  - fasp\_solver\_dbsr\_pbcgs, 292
  - fasp\_solver\_dcsr\_pbcgs, 293
  - fasp\_solver\_dstr\_pbcgs, 293
- pbcgs\_mf.c, 294
  - fasp\_solver\_pbcgs, 295
- pcg.c, 296
  - fasp\_solver\_bdcsr\_pcg, 298
  - fasp\_solver\_dbsr\_pcg, 299
  - fasp\_solver\_dcsr\_pcg, 299
  - fasp\_solver\_dstr\_pcg, 301
- pcg\_mf.c, 302
  - fasp\_solver\_pcg, 303
- pcgrid2d
  - fasp.h, 165
- pdiri
  - grid2d, 37
- perf\_idx
  - precond\_FASP\_blkoi\_data, 69
  - precond\_block\_reservoir\_data, 60
- perf\_neigh
  - precond\_FASP\_blkoi\_data, 70
- pfather
  - grid2d, 38
- pgcg.c, 304
  - fasp\_solver\_dcsr\_pgcg, 304
- pgcg\_mf.c, 305
  - fasp\_solver\_pgcg, 305
- pgcr.c, 306
  - fasp\_solver\_dcsr\_pgcr, 307
  - fasp\_solver\_dcsr\_pgcr1, 308
- pgmres.c, 309
  - fasp\_solver\_bdcsr\_pgmres, 309
  - fasp\_solver\_dbsr\_pgmres, 310
  - fasp\_solver\_dcsr\_pgmres, 311
  - fasp\_solver\_dstr\_pgmres, 311
- pgmres\_mf.c, 312

fasp\_solver\_pgmres, 313  
 pgrid2d  
   fasp.h, 165  
 pivot  
   precond\_FASP\_blkoi\_data, 70  
   precond\_block\_reservoir\_data, 60  
 pivot\_S  
   precond\_FASP\_blkoi\_data, 70  
 pivotS  
   precond\_block\_reservoir\_data, 60  
 pminres.c, 313  
   fasp\_solver\_bdcscr\_pminres, 315  
   fasp\_solver\_dcsr\_pminres, 315  
   fasp\_solver\_dstr\_pminres, 316  
 pminres\_mf.c, 317  
   fasp\_solver\_pminres, 318  
 precondition, 56  
 precondition\_FASP\_blkoi\_data, 67  
   A, 69  
   diaginv, 69  
   diaginv\_S, 69  
   diaginv\_noscale, 69  
   maxit, 69  
   mgl\_data, 69  
   neigh, 69  
   order, 69  
   PP, 70  
   perf\_idx, 69  
   perf\_neigh, 70  
   pivot, 70  
   pivot\_S, 70  
   r, 70  
   RR, 70  
   restart, 70  
   SS, 71  
   scaled, 70  
   tol, 71  
   w, 71  
   WW, 71  
 precondition\_bcsr.c, 319  
   fasp\_precond\_block\_SGS\_3, 323  
   fasp\_precond\_block\_SGS\_3\_amg, 325  
   fasp\_precond\_block\_diag\_3, 320  
   fasp\_precond\_block\_diag\_3\_amg, 321  
   fasp\_precond\_block\_diag\_4, 321  
   fasp\_precond\_block\_lower\_3, 321  
   fasp\_precond\_block\_lower\_3\_amg, 323  
   fasp\_precond\_block\_lower\_4, 323  
   fasp\_precond\_block\_upper\_3, 325  
   fasp\_precond\_block\_upper\_3\_amg, 325  
   fasp\_precond\_sweeping, 327  
 precondition\_block\_data, 56  
   A\_diag, 57  
   Abcsr, 57  
   amgparam, 57  
   LU\_diag, 57  
   mgl, 57  
   r, 57  
   precond\_block\_reservoir\_data, 58  
   diag, 59  
   diaginv, 59  
   diaginvS, 60  
   fasp\_block.h, 169  
   order, 60  
   PP, 60  
   perf\_idx, 60  
   pivot, 60  
   pivotS, 60  
   r, 60  
   RR, 60  
   SS, 61  
   scaled, 60  
   w, 61  
   WW, 61  
 precondition\_bsr.c, 327  
   fasp\_precond\_dbsr\_amg, 328  
   fasp\_precond\_dbsr\_amg\_nk, 328  
   fasp\_precond\_dbsr\_diag, 329  
   fasp\_precond\_dbsr\_diag\_nc2, 329  
   fasp\_precond\_dbsr\_diag\_nc3, 330  
   fasp\_precond\_dbsr\_diag\_nc5, 330  
   fasp\_precond\_dbsr\_diag\_nc7, 332  
   fasp\_precond\_dbsr\_ilu, 332  
   fasp\_precond\_dbsr\_nl\_amli, 334  
 precondition\_csr.c, 334  
   fasp\_precond\_Schwarz, 339  
   fasp\_precond\_amg, 335  
   fasp\_precond\_amg\_nk, 336  
   fasp\_precond\_amli, 336  
   fasp\_precond\_diag, 336  
   fasp\_precond\_famg, 337  
   fasp\_precond\_free, 337  
   fasp\_precond\_ilu, 337  
   fasp\_precond\_ilu\_backward, 338  
   fasp\_precond\_ilu\_forward, 338  
   fasp\_precond\_nl\_amli, 339  
   fasp\_precond\_setup, 339  
 precondition\_data, 61  
 precondition\_data\_bsr, 63  
 precondition\_data\_str, 64  
 precondition\_diagbsr, 66  
 precondition\_diagstr, 66  
 precondition\_str.c, 340  
   fasp\_precond\_dstr\_blockgs, 341  
   fasp\_precond\_dstr\_diag, 341  
   fasp\_precond\_dstr\_ilu0, 341  
   fasp\_precond\_dstr\_ilu0\_backward, 342  
   fasp\_precond\_dstr\_ilu0\_forward, 342

- fasp\_precond\_dstr\_ilu1, 342
  - fasp\_precond\_dstr\_ilu1\_backward, 344
  - fasp\_precond\_dstr\_ilu1\_forward, 344
- precond\_sweeping\_data, 71
  - A, 72
  - Ai, 72
  - local\_A, 72
  - local\_LU, 72
  - local\_index, 72
  - NumLayers, 72
  - r, 73
  - w, 73
- precond\_type
  - input\_param, 48
  - itsolver\_param, 50
- print\_amgcomplexity
  - message.c, 271
- print\_amgcomplexity\_bsr
  - message.c, 271
- print\_cputime
  - message.c, 271
- print\_itinfo
  - message.c, 272
- print\_level
  - input\_param, 48
  - itsolver\_param, 50
- print\_message
  - message.c, 272
- problem\_num
  - input\_param, 48
- pvfgmres.c, 345
  - fasp\_solver\_bdcsr\_pvfgmres, 345
  - fasp\_solver\_dbsr\_pvfgmres, 346
  - fasp\_solver\_dcsr\_pvfgmres, 347
- pvfgmres\_mf.c, 347
  - fasp\_solver\_pvfgmres, 348
- pvgmres.c, 349
  - fasp\_solver\_bdcsr\_pvgmres, 349
  - fasp\_solver\_dbsr\_pvgmres, 350
  - fasp\_solver\_dcsr\_pvgmres, 350
  - fasp\_solver\_dstr\_pvgmres, 352
- pvgmres\_mf.c, 353
  - fasp\_solver\_pvgmres, 353
- quadrature.c, 354
  - fasp\_gauss2d, 354
  - fasp\_quad2d, 355
- r
  - precond\_FASP\_blkoi\_data, 70
  - precond\_block\_data, 57
  - precond\_block\_reservoir\_data, 60
  - precond\_sweeping\_data, 73
- REAL
  - fasp.h, 163
- RR
  - precond\_FASP\_blkoi\_data, 70
  - precond\_block\_reservoir\_data, 60
- RS\_C1
  - fasp.h, 163
- rap.c, 355
  - fasp\_blas\_dcsr\_rap2, 356
- restart
  - input\_param, 48
  - itsolver\_param, 50
  - precond\_FASP\_blkoi\_data, 70
- s
  - grid2d, 38
- SA\_AMG
  - fasp\_const.h, 182
- SCHWARZ\_BACKWARD
  - fasp\_const.h, 183
- SCHWARZ\_FORWARD
  - fasp\_const.h, 183
- SCHWARZ\_SYMMETRIC
  - fasp\_const.h, 183
- SHORT
  - fasp.h, 164
- SMALLREAL
  - fasp\_const.h, 183
- SMALLREAL2
  - fasp\_const.h, 183
- SMOOTHER\_BLKOil
  - fasp\_block.h, 168
- SMOOTHER\_CG
  - fasp\_const.h, 183
- SMOOTHER\_GS
  - fasp\_const.h, 183
- SMOOTHER\_GSOR
  - fasp\_const.h, 183
- SMOOTHER\_JACOBI
  - fasp\_const.h, 184
- SMOOTHER\_L1DIAG
  - fasp\_const.h, 184
- SMOOTHER\_POLY
  - fasp\_const.h, 184
- SMOOTHER\_SGS
  - fasp\_const.h, 184
- SMOOTHER\_SGSOR
  - fasp\_const.h, 184
- SMOOTHER\_SOR
  - fasp\_const.h, 184
- SMOOTHER\_SPETEN
  - fasp\_block.h, 168
- SMOOTHER\_SSOR
  - fasp\_const.h, 184
- SOLVER\_AMG
  - fasp\_const.h, 184

- SOLVER\_BiCGstab
  - fasp\_const.h, [185](#)
- SOLVER\_CG
  - fasp\_const.h, [185](#)
- SOLVER\_DEFAULT
  - fasp\_const.h, [185](#)
- SOLVER\_FMG
  - fasp\_const.h, [185](#)
- SOLVER\_GCG
  - fasp\_const.h, [185](#)
- SOLVER\_GCR
  - fasp\_const.h, [185](#)
- SOLVER\_GMRES
  - fasp\_const.h, [185](#)
- SOLVER\_MUMPS
  - fasp\_const.h, [186](#)
- SOLVER\_MinRes
  - fasp\_const.h, [185](#)
- SOLVER\_SBiCGstab
  - fasp\_const.h, [186](#)
- SOLVER\_SCG
  - fasp\_const.h, [186](#)
- SOLVER\_SGCG
  - fasp\_const.h, [186](#)
- SOLVER\_SGMRES
  - fasp\_const.h, [186](#)
- SOLVER\_SMinRes
  - fasp\_const.h, [186](#)
- SOLVER\_SUPERLU
  - fasp\_const.h, [186](#)
- SOLVER\_SVFGMRES
  - fasp\_const.h, [186](#)
- SOLVER\_SVGMRES
  - fasp\_const.h, [186](#)
- SOLVER\_UMFPACK
  - fasp\_const.h, [187](#)
- SOLVER\_VFGMRES
  - fasp\_const.h, [187](#)
- SOLVER\_VGMRES
  - fasp\_const.h, [187](#)
- SS
  - precond\_FASP\_blkoi\_data, [71](#)
  - precond\_block\_reservoir\_data, [61](#)
- STAG\_RATIO
  - fasp\_const.h, [187](#)
- STOP\_MOD\_REL\_RES
  - fasp\_const.h, [187](#)
- STOP\_REL\_PRECRES
  - fasp\_const.h, [187](#)
- STOP\_REL\_RES
  - fasp\_const.h, [187](#)
- SWAP
  - smat.c, [360](#)
- scaled
  - precond\_FASP\_blkoi\_data, [70](#)
  - precond\_block\_reservoir\_data, [60](#)
- Schwarz\_blksolver
  - input\_param, [48](#)
- Schwarz\_data, [73](#)
- Schwarz\_maxlvl
  - input\_param, [48](#)
- Schwarz\_mmsize
  - input\_param, [48](#)
- Schwarz\_param, [74](#)
- schwarz\_setup.c, [356](#)
  - fasp\_Schwarz\_get\_block\_matrix, [358](#)
  - fasp\_Schwarz\_setup, [358](#)
  - fasp\_dcsr\_Schwarz\_backward\_smoother, [357](#)
  - fasp\_dcsr\_Schwarz\_forward\_smoother, [357](#)
- Schwarz\_type
  - input\_param, [48](#)
- smat.c, [359](#)
  - fasp\_blas\_smat\_Linfinity, [362](#)
  - fasp\_blas\_smat\_inv, [360](#)
  - fasp\_blas\_smat\_inv\_nc, [360](#)
  - fasp\_blas\_smat\_inv\_nc2, [360](#)
  - fasp\_blas\_smat\_inv\_nc3, [361](#)
  - fasp\_blas\_smat\_inv\_nc4, [361](#)
  - fasp\_blas\_smat\_inv\_nc5, [361](#)
  - fasp\_blas\_smat\_inv\_nc7, [362](#)
  - fasp\_blas\_smat\_invp\_nc, [362](#)
  - fasp\_iden\_free, [363](#)
  - fasp\_smat\_identity, [363](#)
  - fasp\_smat\_identity\_nc2, [363](#)
  - fasp\_smat\_identity\_nc3, [365](#)
  - fasp\_smat\_identity\_nc5, [365](#)
  - fasp\_smat\_identity\_nc7, [365](#)
- SWAP, [360](#)
- smoother\_bsr.c, [366](#)
  - fasp\_smoother\_dbsr\_gs, [367](#)
  - fasp\_smoother\_dbsr\_gs1, [367](#)
  - fasp\_smoother\_dbsr\_gs\_ascend, [368](#)
  - fasp\_smoother\_dbsr\_gs\_ascend1, [368](#)
  - fasp\_smoother\_dbsr\_gs\_descend, [369](#)
  - fasp\_smoother\_dbsr\_gs\_descend1, [369](#)
  - fasp\_smoother\_dbsr\_gs\_order1, [370](#)
  - fasp\_smoother\_dbsr\_gs\_order2, [370](#)
  - fasp\_smoother\_dbsr\_ilu, [371](#)
  - fasp\_smoother\_dbsr\_jacobi, [371](#)
  - fasp\_smoother\_dbsr\_jacobi1, [371](#)
  - fasp\_smoother\_dbsr\_jacobi\_setup, [373](#)
  - fasp\_smoother\_dbsr\_sor, [373](#)
  - fasp\_smoother\_dbsr\_sor1, [374](#)
  - fasp\_smoother\_dbsr\_sor\_ascend, [374](#)
  - fasp\_smoother\_dbsr\_sor\_descend, [375](#)
  - fasp\_smoother\_dbsr\_sor\_order, [375](#)
- smoother\_csr.c, [376](#)
  - fasp\_smoother\_dcsr\_L1diag, [380](#)

- fasp\_smoother\_dcsr\_gs, 377
  - fasp\_smoother\_dcsr\_gs\_cf, 377
  - fasp\_smoother\_dcsr\_gs\_rb3d, 378
  - fasp\_smoother\_dcsr\_ilu, 378
  - fasp\_smoother\_dcsr\_jacobi, 379
  - fasp\_smoother\_dcsr\_kaczmarz, 379
  - fasp\_smoother\_dcsr\_sgs, 380
  - fasp\_smoother\_dcsr\_sor, 381
  - fasp\_smoother\_dcsr\_sor\_cf, 381
- smoother\_csr.c, 382
  - fasp\_smoother\_dcsr\_gscr, 382
- smoother\_csr\_poly.c, 383
  - fasp\_smoother\_dcsr\_poly, 384
  - fasp\_smoother\_dcsr\_poly\_old, 384
- smoother\_str.c, 384
  - fasp\_generate\_diaginv\_block, 386
  - fasp\_smoother\_dstr\_gs, 386
  - fasp\_smoother\_dstr\_gs1, 386
  - fasp\_smoother\_dstr\_gs\_ascend, 387
  - fasp\_smoother\_dstr\_gs\_cf, 387
  - fasp\_smoother\_dstr\_gs\_descend, 388
  - fasp\_smoother\_dstr\_gs\_order, 388
  - fasp\_smoother\_dstr\_jacobi, 389
  - fasp\_smoother\_dstr\_jacobi1, 389
  - fasp\_smoother\_dstr\_schwarz, 390
  - fasp\_smoother\_dstr\_sor, 390
  - fasp\_smoother\_dstr\_sor1, 390
  - fasp\_smoother\_dstr\_sor\_ascend, 391
  - fasp\_smoother\_dstr\_sor\_cf, 391
  - fasp\_smoother\_dstr\_sor\_descend, 392
  - fasp\_smoother\_dstr\_sor\_order, 392
- solver\_type
  - input\_param, 48
- sparse\_block.c, 393
  - fasp\_bdcsr\_free, 394
  - fasp\_dbsr\_Linfinity\_dcsr, 396
  - fasp\_dbsr\_getblk, 395
  - fasp\_dbsr\_getblk\_dcsr, 395
  - fasp\_dcsr\_getblk, 396
- sparse\_bsr.c, 397
  - fasp\_dbsr\_alloc, 398
  - fasp\_dbsr\_cp, 398
  - fasp\_dbsr\_create, 399
  - fasp\_dbsr\_diagLU, 401
  - fasp\_dbsr\_diagLU2, 402
  - fasp\_dbsr\_diaginv, 399
  - fasp\_dbsr\_diaginv2, 400
  - fasp\_dbsr\_diaginv3, 400
  - fasp\_dbsr\_diaginv4, 401
  - fasp\_dbsr\_diagpref, 402
  - fasp\_dbsr\_free, 403
  - fasp\_dbsr\_getdiag, 403
  - fasp\_dbsr\_getdiaginv, 404
  - fasp\_dbsr\_null, 404
  - fasp\_dbsr\_trans, 404
- sparse\_coo.c, 405
  - fasp\_dcoo\_alloc, 405
  - fasp\_dcoo\_create, 406
  - fasp\_dcoo\_free, 406
  - fasp\_dcoo\_shift, 406
- sparse\_csr.c, 407
  - fasp\_dcsr\_alloc, 408
  - fasp\_dcsr\_compress, 409
  - fasp\_dcsr\_compress\_inplace, 409
  - fasp\_dcsr\_cp, 410
  - fasp\_dcsr\_create, 410
  - fasp\_dcsr\_diagpref, 410
  - fasp\_dcsr\_free, 412
  - fasp\_dcsr\_getcol, 412
  - fasp\_dcsr\_getdiag, 413
  - fasp\_dcsr\_multicoloring, 413
  - fasp\_dcsr\_null, 414
  - fasp\_dcsr\_perm, 414
  - fasp\_dcsr\_permz, 414
  - fasp\_dcsr\_regdiag, 415
  - fasp\_dcsr\_shift, 415
  - fasp\_dcsr\_sort, 416
  - fasp\_dcsr\_sortz, 416
  - fasp\_dcsr\_symdiagscale, 416
  - fasp\_dcsr\_sympat, 417
  - fasp\_dcsr\_trans, 417
  - fasp\_dcsr\_transz, 418
  - fasp\_icsr\_cp, 418
  - fasp\_icsr\_create, 419
  - fasp\_icsr\_free, 419
  - fasp\_icsr\_null, 419
  - fasp\_icsr\_trans, 421
- sparse\_csrl.c, 421
  - fasp\_dcsrl\_create, 422
  - fasp\_dcsrl\_free, 422
- sparse\_str.c, 422
  - fasp\_dstr\_alloc, 423
  - fasp\_dstr\_cp, 423
  - fasp\_dstr\_create, 425
  - fasp\_dstr\_free, 425
  - fasp\_dstr\_null, 426
- sparse\_util.c, 426
  - fasp\_sparse\_MIS, 430
  - fasp\_sparse\_aat\_, 427
  - fasp\_sparse\_abyb\_, 428
  - fasp\_sparse\_abybms\_, 428
  - fasp\_sparse\_aplbms\_, 429
  - fasp\_sparse\_aplusb\_, 429
  - fasp\_sparse\_iit\_, 429
  - fasp\_sparse\_rapcmp\_, 430
  - fasp\_sparse\_rapms\_, 431
  - fasp\_sparse\_wta\_, 431
  - fasp\_sparse\_wtams\_, 432

- fasp\_sparse\_ytx\_, 432
  - fasp\_sparse\_ytxbig\_, 433
- spbcgs.c, 433
  - fasp\_solver\_bdcscr\_spbcgs, 435
  - fasp\_solver\_dbsr\_spbcgs, 435
  - fasp\_solver\_dcsr\_spbcgs, 436
  - fasp\_solver\_dstr\_spbcgs, 436
- spcg.c, 438
  - fasp\_solver\_bdcscr\_spcg, 440
  - fasp\_solver\_dcsr\_spcg, 440
  - fasp\_solver\_dstr\_spcg, 441
- spgmres.c, 442
  - fasp\_solver\_bdcscr\_spgmres, 442
  - fasp\_solver\_dbsr\_spgmres, 443
  - fasp\_solver\_dcsr\_spgmres, 443
  - fasp\_solver\_dstr\_spgmres, 445
- spminres.c, 446
  - fasp\_solver\_bdcscr\_spminres, 447
  - fasp\_solver\_dcsr\_spminres, 448
  - fasp\_solver\_dstr\_spminres, 448
- spvgmres.c, 450
  - fasp\_solver\_bdcscr\_spvgmres, 451
  - fasp\_solver\_dbsr\_spvgmres, 451
  - fasp\_solver\_dcsr\_spvgmres, 452
  - fasp\_solver\_dstr\_spvgmres, 452
- stop\_type
  - input\_param, 49
  - itsolver\_param, 50
- t
  - grid2d, 38
- THDs\_AMG\_GS
  - threads.c, 454
- THDs\_CPR\_gGS
  - threads.c, 454
- THDs\_CPR\_IGS
  - threads.c, 455
- TRUE
  - fasp\_const.h, 187
- tfather
  - grid2d, 38
- threads.c, 453
  - FASP\_GET\_START\_END, 454
  - fasp\_set\_GS\_threads, 454
  - THDs\_AMG\_GS, 454
  - THDs\_CPR\_gGS, 454
  - THDs\_CPR\_IGS, 455
- timing.c, 455
  - fasp\_gettime, 455
- tol
  - itsolver\_param, 50
  - precond\_FASP\_blkoi\_data, 71
- total\_alloc\_count
  - fasp.h, 166
- total\_alloc\_mem
  - fasp.h, 166
  - memory.c, 269
- triangles
  - grid2d, 38
- UA\_AMG
  - fasp\_const.h, 188
- UNPT
  - fasp\_const.h, 188
- USERDEFINED
  - fasp\_const.h, 188
- V\_CYCLE
  - fasp\_const.h, 188
- VMB
  - fasp\_const.h, 188
- val
  - dBSRmat, 32
- vec.c, 456
  - fasp\_dvec\_alloc, 457
  - fasp\_dvec\_cp, 458
  - fasp\_dvec\_create, 458
  - fasp\_dvec\_free, 458
  - fasp\_dvec\_isnan, 460
  - fasp\_dvec\_maxdiff, 460
  - fasp\_dvec\_null, 461
  - fasp\_dvec\_rand, 461
  - fasp\_dvec\_set, 462
  - fasp\_dvec\_symdiagscale, 462
  - fasp\_ivec\_alloc, 462
  - fasp\_ivec\_create, 463
  - fasp\_ivec\_free, 463
  - fasp\_ivec\_set, 463
- vertices
  - grid2d, 38
- w
  - precond\_FASP\_blkoi\_data, 71
  - precond\_block\_reservoir\_data, 61
  - precond\_sweeping\_data, 73
- W\_CYCLE
  - fasp\_const.h, 188
- WW
  - precond\_FASP\_blkoi\_data, 71
  - precond\_block\_reservoir\_data, 61
- workdir
  - input\_param, 49
- wrapper.c, 464
  - fasp\_fwapper\_amg\_, 465
  - fasp\_fwapper\_krylov\_amg\_, 465
  - fasp\_wrapper\_dbsr\_krylov\_amg, 466
  - fasp\_wrapper\_dcoo\_dbsr\_krylov\_amg, 466