

Fast Auxiliary Space Preconditioning

1.8.4 Feb/15/2016

Generated by Doxygen 1.8.10

Tue Feb 16 2016 09:19:20

Contents

1	Introduction	1
2	How to obtain FASP	3
3	Building and Installation	5
4	Developers	7
5	Doxygen	9
6	Todo List	11
7	Data Structure Index	13
7.1	Data Structures	13
8	File Index	17
8.1	File List	17
9	Data Structure Documentation	23
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	26
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	29
9.5	block_dCSRmat Struct Reference	29
9.5.1	Detailed Description	29
9.6	block_dvector Struct Reference	29
9.6.1	Detailed Description	30
9.7	block_iCSRmat Struct Reference	30

9.7.1 Detailed Description	30
9.8 block_ivector Struct Reference	30
9.8.1 Detailed Description	31
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	33
9.11.1 Detailed Description	33
9.12 dCSRmat Struct Reference	33
9.12.1 Detailed Description	34
9.13 dCSRmat Struct Reference	34
9.13.1 Detailed Description	35
9.14 ddenmat Struct Reference	35
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	37
9.17 grid2d Struct Reference	37
9.17.1 Detailed Description	37
9.17.2 Field Documentation	37
9.17.2.1 e	37
9.17.2.2 edges	38
9.17.2.3 ediri	38
9.17.2.4 efather	38
9.17.2.5 p	38
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	39
9.17.2.12 vertices	39

9.18 iCOOmat Struct Reference	39
9.18.1 Detailed Description	39
9.19 iCSRmat Struct Reference	40
9.19.1 Detailed Description	40
9.20 idenmat Struct Reference	40
9.20.1 Detailed Description	41
9.21 ILU_data Struct Reference	41
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	44
9.23.2.3 AMG_aggressive_path	44
9.23.2.4 AMG_amli_degree	44
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	45
9.23.2.12 AMG_levels	45
9.23.2.13 AMG_max_aggregation	45
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	46
9.23.2.21 AMG_quality_bound	46
9.23.2.22 AMG_relaxation	46
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	47
9.23.2.30	AMG_tol	47
9.23.2.31	AMG_truncation_threshold	47
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	48
9.23.2.39	itsolver_maxit	48
9.23.2.40	itsolver_tol	48
9.23.2.41	output_type	48
9.23.2.42	precond_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_blk solver	48
9.23.2.47	Schwarz_maxlvl	49
9.23.2.48	Schwarz_mmsize	49
9.23.2.49	Schwarz_type	49
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	50
9.24.2	Field Documentation	50
9.24.2.1	itsolver_type	50
9.24.2.2	maxit	50
9.24.2.3	precond_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	51
9.25	ivector Struct Reference	51
9.25.1	Detailed Description	51
9.26	Link Struct Reference	51
9.26.1	Detailed Description	52
9.27	linked_list Struct Reference	52
9.27.1	Detailed Description	52
9.28	mallinfo Struct Reference	52
9.28.1	Detailed Description	53
9.29	malloc_chunk Struct Reference	53
9.29.1	Detailed Description	53
9.30	malloc_params Struct Reference	53
9.30.1	Detailed Description	54
9.31	malloc_segment Struct Reference	54
9.31.1	Detailed Description	54
9.32	malloc_state Struct Reference	54
9.32.1	Detailed Description	55
9.33	malloc_tree_chunk Struct Reference	55
9.33.1	Detailed Description	55
9.34	Mumps_data Struct Reference	55
9.34.1	Detailed Description	55
9.35	mxv_matfree Struct Reference	56
9.35.1	Detailed Description	56
9.36	nedmallinfo Struct Reference	56
9.36.1	Detailed Description	56
9.37	Pardiso_data Struct Reference	56
9.37.1	Detailed Description	57
9.38	precond Struct Reference	57
9.38.1	Detailed Description	57
9.39	precond_block_data Struct Reference	57
9.39.1	Detailed Description	58
9.39.2	Field Documentation	58
9.39.2.1	A_diag	58
9.39.2.2	Abcsr	58
9.39.2.3	amgparam	58
9.39.2.4	LU_diag	58
9.39.2.5	mgl	58

9.39.2.6	r	59
9.40	precond_block_reservoir_data Struct Reference	59
9.40.1	Detailed Description	60
9.40.2	Field Documentation	60
9.40.2.1	diag	60
9.40.2.2	diaginv	61
9.40.2.3	diaginvS	61
9.40.2.4	order	61
9.40.2.5	perf_idx	61
9.40.2.6	pivot	61
9.40.2.7	pivotS	61
9.40.2.8	PP	61
9.40.2.9	r	61
9.40.2.10	RR	61
9.40.2.11	scaled	62
9.40.2.12	SS	62
9.40.2.13	w	62
9.40.2.14	WW	62
9.41	precond_data Struct Reference	62
9.41.1	Detailed Description	63
9.42	precond_data_bsr Struct Reference	64
9.42.1	Detailed Description	65
9.43	precond_data_str Struct Reference	65
9.43.1	Detailed Description	67
9.44	precond_diagbsr Struct Reference	67
9.44.1	Detailed Description	67
9.45	precond_diagstr Struct Reference	67
9.45.1	Detailed Description	68
9.46	precond_FASP_blkcoil_data Struct Reference	68
9.46.1	Detailed Description	69
9.46.2	Field Documentation	70
9.46.2.1	A	70
9.46.2.2	diaginv	70
9.46.2.3	diaginv_noscale	70
9.46.2.4	diaginv_S	70
9.46.2.5	maxit	70
9.46.2.6	mgl_data	70

9.46.2.7	neigh	70
9.46.2.8	order	70
9.46.2.9	perf_idx	71
9.46.2.10	perf_neigh	71
9.46.2.11	pivot	71
9.46.2.12	pivot_S	71
9.46.2.13	PP	71
9.46.2.14	r	71
9.46.2.15	restart	71
9.46.2.16	RR	71
9.46.2.17	scaled	71
9.46.2.18	SS	72
9.46.2.19	tol	72
9.46.2.20	w	72
9.46.2.21	WW	72
9.47	precond_sweeping_data Struct Reference	72
9.47.1	Detailed Description	73
9.47.2	Field Documentation	73
9.47.2.1	A	73
9.47.2.2	Ai	73
9.47.2.3	local_A	73
9.47.2.4	local_index	73
9.47.2.5	local_LU	73
9.47.2.6	NumLayers	74
9.47.2.7	r	74
9.47.2.8	w	74
9.48	Schwarz_data Struct Reference	74
9.48.1	Detailed Description	75
9.49	Schwarz_param Struct Reference	75
9.49.1	Detailed Description	76
10	File Documentation	77
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 fasp_amg_setup_cr(AMG_data *mgl, AMG_param *param)	78
10.3 amg_setup_rs.c File Reference	79
10.3.1 Detailed Description	79
10.3.2 Function Documentation	79
10.3.2.1 fasp_amg_setup_rs(AMG_data *mgl, AMG_param *param)	79
10.4 amg_setup_sa.c File Reference	80
10.4.1 Detailed Description	80
10.4.2 Function Documentation	81
10.4.2.1 fasp_amg_setup_sa(AMG_data *mgl, AMG_param *param)	81
10.4.2.2 fasp_amg_setup_sa_bsr(AMG_data_bsr *mgl, AMG_param *param)	81
10.5 amg_setup_ua.c File Reference	82
10.5.1 Detailed Description	82
10.5.2 Function Documentation	82
10.5.2.1 fasp_amg_setup_ua(AMG_data *mgl, AMG_param *param)	82
10.5.2.2 fasp_amg_setup_ua_bsr(AMG_data_bsr *mgl, AMG_param *param)	83
10.6 amg_solve.c File Reference	84
10.6.1 Detailed Description	84
10.6.2 Function Documentation	85
10.6.2.1 fasp_amg_solve(AMG_data *mgl, AMG_param *param)	85
10.6.2.2 fasp_amg_solve_amli(AMG_data *mgl, AMG_param *param)	86
10.6.2.3 fasp_amg_solve_nl_amli(AMG_data *mgl, AMG_param *param)	86
10.6.2.4 fasp_famg_solve(AMG_data *mgl, AMG_param *param)	87
10.7 amlirecur.c File Reference	87
10.7.1 Detailed Description	88
10.7.2 Function Documentation	88
10.7.2.1 fasp_amg_amli_coef(const REAL lambda_max, const REAL lambda_min, const INT degree, REAL *coef)	88
10.7.2.2 fasp_solver_amli(AMG_data *mgl, AMG_param *param, INT level)	88
10.7.2.3 fasp_solver_nl_amli(AMG_data *mgl, AMG_param *param, INT level, INT num_levels)	89
10.7.2.4 fasp_solver_nl_amli_bsr(AMG_data_bsr *mgl, AMG_param *param, INT level, INT num_levels)	89
10.8 array.c File Reference	90
10.8.1 Detailed Description	91
10.8.2 Function Documentation	91
10.8.2.1 fasp_array_cp(const INT n, REAL *x, REAL *y)	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_aAxpby(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 fasp_blas_dbsr_rap1(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)	107
10.11.2.10 fasp_blas_dbsr_rap_agg(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)	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_m xv(dCSRmat *A, REAL *x, REAL *y)	112
10.12.2.8 fasp_blas_dcsr_m xv_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_dcsl_m xv(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_aApyb(const REAL alpha, REAL *A, REAL *x, const REAL beta, REAL *y, const INT n)	124

10.14.2.10	<code>fasp_blas_smat_add</code> (REAL *a, REAL *b, const INT n, const REAL alpha, const REAL beta, REAL *c)	124
10.14.2.11	<code>fasp_blas_smat_axm</code> (REAL *a, const INT n, const REAL alpha)	125
10.14.2.12	<code>fasp_blas_smat_mul</code> (REAL *a, REAL *b, REAL *c, const INT n)	125
10.14.2.13	<code>fasp_blas_smat_mul_nc2</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.14	<code>fasp_blas_smat_mul_nc3</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.15	<code>fasp_blas_smat_mul_nc5</code> (REAL *a, REAL *b, REAL *c)	126
10.14.2.16	<code>fasp_blas_smat_mul_nc7</code> (REAL *a, REAL *b, REAL *c)	127
10.14.2.17	<code>fasp_blas_smat_m xv</code> (REAL *a, REAL *b, REAL *c, const INT n)	127
10.14.2.18	<code>fasp_blas_smat_m xv_nc2</code> (REAL *a, REAL *b, REAL *c)	128
10.14.2.19	<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 fasp_blas_dvec_axpy(const REAL a, dvector *x, dvector *y)	141
10.16.2.2 fasp_blas_dvec_axpyz(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, iverector *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, iverector *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 dvector	164
10.24.3.7 grid2d	164
10.24.3.8 iCOOmat	164
10.24.3.9 iCSRmat	165
10.24.3.10 iddenmat	165
10.24.3.11 ivector	165
10.24.3.12 linkList	165
10.24.3.13 listElement	165
10.24.3.14 pcgrid2d	165
10.24.3.15 pgrid2d	165
10.24.4 Variable Documentation	165
10.24.4.1 count	165
10.24.4.2 IMAP	165
10.24.4.3 MAXIMAP	165
10.24.4.4 nx_rb	166
10.24.4.5 ny_rb	166
10.24.4.6 nz_rb	166
10.24.4.7 total_alloc_count	166
10.24.4.8 total_alloc_mem	166
10.25 fasp_block.h File Reference	166
10.25.1 Detailed Description	167
10.25.2 Macro Definition Documentation	168
10.25.2.1 __FASPBLOCK_HEADER__	168
10.25.2.2 SMOOTHER_BLKOIL	168
10.25.2.3 SMOOTHER_SPETEN	168
10.25.3 Typedef Documentation	168
10.25.3.1 block_BSR	168
10.25.3.2 block_dCSRmat	168
10.25.3.3 block_dvector	168
10.25.3.4 block_iCSRmat	168
10.25.3.5 block_ivecior	168
10.25.3.6 block_Reservoir	168
10.25.3.7 dBSRmat	169
10.25.3.8 precondition_block_reservoir_data	169
10.26 fasp_const.h File Reference	169

10.26.1 Detailed Description	172
10.26.2 Macro Definition Documentation	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	SPT	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

10.26.2.11	S SOLVER_GMRES	185
10.26.2.11	S SOLVER_MinRes	186
10.26.2.11	S SOLVER_MUMPS	186
10.26.2.11	S SOLVER_PARDISO	186
10.26.2.11	S SOLVER_SBiCGstab	186
10.26.2.11	S SOLVER_SCG	186
10.26.2.11	S SOLVER_SGCG	186
10.26.2.11	S SOLVER_SGMRES	186
10.26.2.11	S SOLVER_SMinRes	186
10.26.2.11	S SOLVER_SUPERLU	186
10.26.2.12	S SOLVER_SVFGMRES	187
10.26.2.12	S SOLVER_SVGMRES	187
10.26.2.12	S SOLVER_UMFPACK	187
10.26.2.12	S SOLVER_VFGMRES	187
10.26.2.12	S SOLVER_VGMRES	187
10.26.2.12	S STAG_RATIO	187
10.26.2.12	S STOP_MOD_REL_RES	187
10.26.2.12	S STOP_REL_PRECRES	187
10.26.2.12	S STOP_REL_RES	187
10.26.2.12	S TRUE	188
10.26.2.13	S UA_AMG	188
10.26.2.13	S UNPT	188
10.26.2.13	S USERDEFINED	188
10.26.2.13	S _CYCLE	188
10.26.2.13	S MMB	188
10.26.2.13	S _CYCLE	188
10.27	fmgcycle.c File Reference	189
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)	191
10.28.2.3	fasp_format_dbsr_dcsr(dBSRmat *B)	191
10.28.2.4	fasp_format_dcoo_dcsr(dCOOmat *A, dCSRmat *B)	192

10.28.2.5 fasp_format_dcsr_dbsr(dCSRmat *A, const INT nb)	193
10.28.2.6 fasp_format_dcsr_dcoo(dCSRmat *A, dCOOmat *B)	193
10.28.2.7 fasp_format_dcsr_dcsr(dCSRmat *A)	194
10.28.2.8 fasp_format_dstr_dbsr(dSTRmat *B)	194
10.28.2.9 fasp_format_dstr_dcsr(dSTRmat *A, dCSRmat *B)	195
10.29givens.c File Reference	195
10.29.1 Detailed Description	196
10.29.2 Function Documentation	196
10.29.2.1 fasp_aux_givens(const REAL beta, dCSRmat *H, dvector *y, REAL *tmp)	196
10.30gmg_poisson.c File Reference	196
10.30.1 Detailed Description	197
10.30.2 Function Documentation	197
10.30.2.1 fasp_poisson_fgmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtvl)	197
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 prtvl)	197
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 prtvl)	198
10.30.2.4 fasp_poisson_gmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtvl)	198
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 prtvl)	199
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 prtvl)	199
10.30.2.7 fasp_poisson_pcg_gmg_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtvl)	200
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 prtvl)	201
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 prtvl)	202
10.31graphics.c File Reference	203
10.31.1 Detailed Description	203
10.31.2 Function Documentation	203
10.31.2.1 fasp_dbsr_plot(const dBSRmat *A, const char *fname)	203
10.31.2.2 fasp_dbsr_subplot(const dBSRmat *A, const char *filename, INT size)	204
10.31.2.3 fasp_dcsr_plot(const dCSRmat *A, const char *fname)	204
10.31.2.4 fasp_dcsr_subplot(const dCSRmat *A, const char *filename, INT size)	205
10.31.2.5 fasp_grid2d_plot(pgrid2d pg, INT level)	205
10.32ilu_setup_bsr.c File Reference	206
10.32.1 Detailed Description	206

10.32.2 Function Documentation	206
10.32.2.1 fasp_ilu_dbsr_setup(dBSRmat *A, ILU_data *iludata, ILU_param *iluparam)	206
10.33ilu_setup_csr.c File Reference	207
10.33.1 Detailed Description	207
10.33.2 Function Documentation	207
10.33.2.1 fasp_ilu_dcsr_setup(dCSRmat *A, ILU_data *iludata, ILU_param *iluparam)	207
10.34ilu_setup_str.c File Reference	208
10.34.1 Detailed Description	208
10.34.2 Function Documentation	208
10.34.2.1 fasp_ilu_dstr_setup0(dSTRmat *A, dSTRmat *LU)	208
10.34.2.2 fasp_ilu_dstr_setup1(dSTRmat *A, dSTRmat *LU)	209
10.35init.c File Reference	209
10.35.1 Detailed Description	210
10.35.2 Function Documentation	210
10.35.2.1 fasp_amg_data_bsr_create(SHORT max_levels)	210
10.35.2.2 fasp_amg_data_bsr_free(AMG_data_bsr *mgl)	211
10.35.2.3 fasp_amg_data_create(SHORT max_levels)	211
10.35.2.4 fasp_amg_data_free(AMG_data *mgl, AMG_param *param)	211
10.35.2.5 fasp_ilu_data_alloc(const INT iwk, const INT nwork, ILU_data *iludata)	212
10.35.2.6 fasp_ilu_data_free(ILU_data *ILUdata)	212
10.35.2.7 fasp_ilu_data_null(ILU_data *ILUdata)	212
10.35.2.8 fasp_precond_data_null(precond_data *pcdata)	213
10.35.2.9 fasp_precond_null(precond *pcdata)	213
10.35.2.10 fasp_Schwarz_data_free(Schwarz_data *Schwarz)	213
10.36input.c File Reference	214
10.36.1 Detailed Description	214
10.36.2 Function Documentation	214
10.36.2.1 fasp_param_check(input_param *inparam)	214
10.36.2.2 fasp_param_input(const char *filenm, input_param *inparam)	215
10.37interface_mumps.c File Reference	215
10.37.1 Detailed Description	216
10.37.2 Macro Definition Documentation	216
10.37.2.1 ICNTL	216
10.37.3 Function Documentation	216
10.37.3.1 fasp_solver_mumps(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)	216
10.37.3.2 fasp_solver_mumps_steps(dCSRmat *ptrA, dvector *b, dvector *u, Mumps_data *mumps)	216

10.38	interface_pardiso.c File Reference	217
10.38.1	Detailed Description	217
10.38.2	Function Documentation	217
10.38.2.1	fasp_solver_pardiso(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)	217
10.39	interface_samg.c File Reference	218
10.39.1	Detailed Description	218
10.39.2	Function Documentation	218
10.39.2.1	dCSRmat2SAMGInput(dCSRmat *A, char *filefrm, char *fileamg)	218
10.39.2.2	dvector2SAMGInput(dvector *vec, char *filename)	218
10.40	interface_superlu.c File Reference	219
10.40.1	Detailed Description	219
10.40.2	Function Documentation	219
10.40.2.1	fasp_solver_superlu(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)	219
10.41	interface_umfpack.c File Reference	220
10.41.1	Detailed Description	220
10.41.2	Function Documentation	220
10.41.2.1	fasp_solver_umfpack(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)	220
10.42	interpolation.c File Reference	221
10.42.1	Detailed Description	221
10.42.2	Function Documentation	221
10.42.2.1	fasp_amg_interp(dCSRmat *A, ivector *vertices, dCSRmat *P, iCSRmat *S, AMG_param *param)	221
10.42.2.2	fasp_amg_interp1(dCSRmat *A, ivector *vertices, dCSRmat *P, AMG_param *param, iCSRmat *S, INT *icor_ysk)	222
10.42.2.3	fasp_amg_interp_trunc(dCSRmat *P, AMG_param *param)	222
10.43	interpolation_em.c File Reference	223
10.43.1	Detailed Description	223
10.43.2	Function Documentation	223
10.43.2.1	fasp_amg_interp_em(dCSRmat *A, ivector *vertices, dCSRmat *P, AMG_param *param)	223
10.44	io.c File Reference	224
10.44.1	Detailed Description	226
10.44.2	Function Documentation	226
10.44.2.1	fasp_dbsr_print(dBSRmat *A)	226
10.44.2.2	fasp_dbsr_read(const char *filename, dBSRmat *A)	226
10.44.2.3	fasp_dbsr_write(const char *filename, dBSRmat *A)	227
10.44.2.4	fasp_dbsr_write_coo(const char *filename, const dBSRmat *A)	227
10.44.2.5	fasp_dcoo1_read(const char *filename, dCOOmat *A)	228

10.44.2.6	<code>fasp_dcoo_print(dCOOmat *A)</code>	228
10.44.2.7	<code>fasp_dcoo_read(const char *filename, dCSRmat *A)</code>	229
10.44.2.8	<code>fasp_dcoo_shift_read(const char *filename, dCSRmat *A)</code>	229
10.44.2.9	<code>fasp_dcoo_write(const char *filename, dCSRmat *A)</code>	230
10.44.2.10	<code>fasp_dcsr_print(dCSRmat *A)</code>	230
10.44.2.11	<code>fasp_dcsr_read(const char *filename, dCSRmat *A)</code>	231
10.44.2.12	<code>fasp_dcsr_write_coo(const char *filename, const dCSRmat *A)</code>	231
10.44.2.13	<code>fasp_dcsrvec1_read(const char *filename, dCSRmat *A, dvector *b)</code>	231
10.44.2.14	<code>fasp_dcsrvec1_write(const char *filename, dCSRmat *A, dvector *b)</code>	232
10.44.2.15	<code>fasp_dcsrvec2_read(const char *filemat, const char *filerhs, dCSRmat *A, dvector *b)</code>	233
10.44.2.16	<code>fasp_dcsrvec2_write(const char *filemat, const char *filerhs, dCSRmat *A, dvector *b)</code>	233
10.44.2.17	<code>fasp_dmtx_read(const char *filename, dCSRmat *A)</code>	234
10.44.2.18	<code>fasp_dmtxsym_read(const char *filename, dCSRmat *A)</code>	235
10.44.2.19	<code>fasp_dstr_print(dSTRmat *A)</code>	236
10.44.2.20	<code>fasp_dstr_read(const char *filename, dSTRmat *A)</code>	236
10.44.2.21	<code>fasp_dstr_write(const char *filename, dSTRmat *A)</code>	237
10.44.2.22	<code>fasp_dvec_print(INT n, dvector *u)</code>	237
10.44.2.23	<code>fasp_dvec_read(const char *filename, dvector *b)</code>	238
10.44.2.24	<code>fasp_dvec_write(const char *filename, dvector *vec)</code>	238
10.44.2.25	<code>fasp_dvecind_read(const char *filename, dvector *b)</code>	239
10.44.2.26	<code>fasp_dvecind_write(const char *filename, dvector *vec)</code>	239
10.44.2.27	<code>fasp_hb_read(const char *input_file, dCSRmat *A, dvector *b)</code>	239
10.44.2.28	<code>fasp_ivec_print(INT n, ivector *u)</code>	240
10.44.2.29	<code>fasp_ivec_read(const char *filename, ivector *b)</code>	240
10.44.2.30	<code>fasp_ivec_write(const char *filename, ivector *vec)</code>	241
10.44.2.31	<code>fasp_ivecind_read(const char *filename, ivector *b)</code>	241
10.44.2.32	<code>fasp_matrix_read(const char *filename, void *A)</code>	242
10.44.2.33	<code>fasp_matrix_read_bin(const char *filename, void *A)</code>	242
10.44.2.34	<code>fasp_matrix_write(const char *filename, void *A, INT flag)</code>	243
10.44.2.35	<code>fasp_vector_read(const char *filerhs, void *b)</code>	243
10.44.2.36	<code>fasp_vector_write(const char *filerhs, void *b, INT flag)</code>	244
10.44.3	Variable Documentation	245
10.44.3.1	<code>dlength</code>	245
10.44.3.2	<code>ilength</code>	245
10.45	<code>itsolver_bcsr.c</code> File Reference	245
10.45.1	Detailed Description	246
10.45.2	Function Documentation	246

10.45.2.1 fasp_solver_bdcsltsolver(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	246
10.45.2.2 fasp_solver_bdcsltsolver_krylov(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	246
10.45.2.3 fasp_solver_bdcsltsolver_krylov_block_3(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)	247
10.45.2.4 fasp_solver_bdcsltsolver_krylov_block_4(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)	247
10.45.2.5 fasp_solver_bdcsltsolver_krylov_sweeping(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, INT NumLayers, block_dCSRmat *Ai, dCSRmat *local_A, ivector *local_index)	248
10.46 itsolver_bsr.c File Reference	249
10.46.1 Detailed Description	249
10.46.2 Function Documentation	249
10.46.2.1 fasp_solver_dbssltsolver(dBSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	249
10.46.2.2 fasp_solver_dbssltsolver_krylov(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	250
10.46.2.3 fasp_solver_dbssltsolver_krylov_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam)	250
10.46.2.4 fasp_solver_dbssltsolver_krylov_amg_nk(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_nk, dCSRmat *P_nk, dCSRmat *R_nk)	251
10.46.2.5 fasp_solver_dbssltsolver_krylov_diag(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	251
10.46.2.6 fasp_solver_dbssltsolver_krylov_ilu(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)	252
10.46.2.7 fasp_solver_dbssltsolver_krylov_nk_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, const INT nk_dim, dvector *nk)	252
10.47 itsolver_csr.c File Reference	253
10.47.1 Detailed Description	254
10.47.2 Function Documentation	254
10.47.2.1 fasp_solver_dcsltsolver(dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	254
10.47.2.2 fasp_solver_dcsltsolver_krylov(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	254
10.47.2.3 fasp_solver_dcsltsolver_krylov_amg(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam)	255
10.47.2.4 fasp_solver_dcsltsolver_krylov_amg_nk(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_nk, dCSRmat *P_nk, dCSRmat *R_nk)	255
10.47.2.5 fasp_solver_dcsltsolver_krylov_diag(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	256
10.47.2.6 fasp_solver_dcsltsolver_krylov_ilu(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)	256

10.47.2.7 fasp_solver_dcsr_krylov_ilu_M(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam, dCSRmat *M)	257
10.47.2.8 fasp_solver_dcsr_krylov_Schwarz(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, Schwarz_param *schparam)	258
10.48 itsolver_mf.c File Reference	259
10.48.1 Detailed Description	259
10.48.2 Function Documentation	260
10.48.2.1 fasp_solver_itsolver(mxv_matfree *mf, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	260
10.48.2.2 fasp_solver_itsolver_init(INT matrix_format, mxv_matfree *mf, void *A)	261
10.48.2.3 fasp_solver_krylov(mxv_matfree *mf, dvector *b, dvector *x, itsolver_param *itparam)	261
10.49 itsolver_str.c File Reference	262
10.49.1 Detailed Description	262
10.49.2 Function Documentation	263
10.49.2.1 fasp_solver_dstr_itsolver(dSTRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)	263
10.49.2.2 fasp_solver_dstr_krylov(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	263
10.49.2.3 fasp_solver_dstr_krylov_blockgs(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ivector *neigh, ivector *order)	264
10.49.2.4 fasp_solver_dstr_krylov_diag(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)	265
10.49.2.5 fasp_solver_dstr_krylov_ilu(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)	265
10.50 lu.c File Reference	266
10.50.1 Detailed Description	266
10.50.2 Function Documentation	266
10.50.2.1 fasp_smat_lu_decomp(REAL *A, INT pivot[], const INT n)	266
10.50.2.2 fasp_smat_lu_solve(REAL *A, REAL b[], INT pivot[], REAL x[], const INT n)	267
10.51 memory.c File Reference	268
10.51.1 Detailed Description	268
10.51.2 Function Documentation	269
10.51.2.1 fasp_mem_calloc(LONGLONG size, INT type)	269
10.51.2.2 fasp_mem_check(void *ptr, const char *message, INT ERR)	269
10.51.2.3 fasp_mem_dcsr_check(dCSRmat *A)	269
10.51.2.4 fasp_mem_free(void *mem)	270
10.51.2.5 fasp_mem_iludata_check(ILU_data *iludata)	270
10.51.2.6 fasp_mem_realloc(void *oldmem, LONGLONG tsize)	271
10.51.2.7 fasp_mem_usage()	271
10.51.3 Variable Documentation	271

10.51.3.1 total_alloc_count	271
10.51.3.2 total_alloc_mem	272
10.52message.c File Reference	272
10.52.1 Detailed Description	272
10.52.2 Function Documentation	272
10.52.2.1 fasp_chkerr(const SHORT status, const char *fctname)	272
10.52.2.2 print_amgcomplexity(AMG_data *mgl, const SHORT prtlvl)	273
10.52.2.3 print_amgcomplexity_bsr(AMG_data_bsr *mgl, const SHORT prtlvl)	273
10.52.2.4 print_cputime(const char *message, const REAL cputime)	273
10.52.2.5 print_itinfo(const INT prtlvl, const INT stop_type, const INT iter, const REAL relres, const REAL absres, const REAL factor)	274
10.52.2.6 print_message(const INT prtlvl, const char *message)	274
10.53mgcycle.c File Reference	275
10.53.1 Detailed Description	275
10.53.2 Function Documentation	275
10.53.2.1 fasp_solver_mgcycle(AMG_data *mgl, AMG_param *param)	275
10.53.2.2 fasp_solver_mgcycle_bsr(AMG_data_bsr *mgl, AMG_param *param)	276
10.54mgrecur.c File Reference	276
10.54.1 Detailed Description	276
10.54.2 Function Documentation	276
10.54.2.1 fasp_solver_mgrecur(AMG_data *mgl, AMG_param *param, INT level)	276
10.55ordering.c File Reference	277
10.55.1 Detailed Description	278
10.55.2 Function Documentation	278
10.55.2.1 fasp_aux_dQuickSort(REAL *a, INT left, INT right)	278
10.55.2.2 fasp_aux_dQuickSortIndex(REAL *a, INT left, INT right, INT *index)	278
10.55.2.3 fasp_aux_iQuickSort(INT *a, INT left, INT right)	279
10.55.2.4 fasp_aux_iQuickSortIndex(INT *a, INT left, INT right, INT *index)	280
10.55.2.5 fasp_aux_merge(INT numbers[], INT work[], INT left, INT mid, INT right)	280
10.55.2.6 fasp_aux_msort(INT numbers[], INT work[], INT left, INT right)	281
10.55.2.7 fasp_aux_unique(INT numbers[], const INT size)	281
10.55.2.8 fasp_BinarySearch(INT *list, const INT value, const INT nlist)	282
10.55.2.9 fasp_dcsr_CMK_order(const dCSRmat *A, INT *order, INT *oindex)	282
10.55.2.10 fasp_dcsr_RCMK_order(const dCSRmat *A, INT *order, INT *oindex, INT *rorder)	283
10.56parameters.c File Reference	283
10.56.1 Detailed Description	284
10.56.2 Function Documentation	284

10.56.2.1 fasp_param_amg_init(AMG_param *amgparam)	284
10.56.2.2 fasp_param_amg_print(AMG_param *param)	285
10.56.2.3 fasp_param_amg_set(AMG_param *param, input_param *iniparam)	285
10.56.2.4 fasp_param_amg_to_prec(precond_data *pcdata, AMG_param *amgparam)	285
10.56.2.5 fasp_param_amg_to_prec_bsr(precond_data_bsr *pcdata, AMG_param *amgparam)	286
10.56.2.6 fasp_param_ilu_init(ILU_param *iluparam)	286
10.56.2.7 fasp_param_ilu_print(ILU_param *param)	286
10.56.2.8 fasp_param_ilu_set(ILU_param *iluparam, input_param *iniparam)	287
10.56.2.9 fasp_param_init(input_param *iniparam, itsolver_param *itsparam, AMG_param *amgparam, ILU_param *iluparam, Schwarz_param *schparam)	287
10.56.2.10 fasp_param_input_init(input_param *iniparam)	288
10.56.2.11 fasp_param_prec_to_amg(AMG_param *amgparam, precondition_data *pcdata)	288
10.56.2.12 fasp_param_prec_to_amg_bsr(AMG_param *amgparam, precondition_data_bsr *pcdata)	288
10.56.2.13 fasp_param_Schwarz_init(Schwarz_param *schparam)	289
10.56.2.14 fasp_param_Schwarz_print(Schwarz_param *param)	290
10.56.2.15 fasp_param_Schwarz_set(Schwarz_param *schparam, input_param *iniparam)	290
10.56.2.16 fasp_param_set(int argc, const char *argv[], input_param *iniparam)	290
10.56.2.17 fasp_param_solver_init(itsolver_param *itsparam)	291
10.56.2.18 fasp_param_solver_print(itsolver_param *param)	291
10.56.2.19 fasp_param_solver_set(itsolver_param *itsparam, input_param *iniparam)	291
10.57 pbcgs.c File Reference	292
10.57.1 Detailed Description	292
10.57.2 Function Documentation	293
10.57.2.1 fasp_solver_bdcsr_pbcgs(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)	293
10.57.2.2 fasp_solver_dbsr_pbcgs(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)	294
10.57.2.3 fasp_solver_dcsr_pbcgs(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)	295
10.57.2.4 fasp_solver_dstr_pbcgs(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)	295
10.58 pbcgs_mf.c File Reference	296
10.58.1 Detailed Description	296
10.58.2 Function Documentation	297
10.58.2.1 fasp_solver_pbcgs(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)	297
10.59 pcg.c File Reference	298
10.59.1 Detailed Description	299
10.59.2 Function Documentation	300

10.59.2.1 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 prtvl)	300
10.59.2.2 fasp_solver_dbsl_pcg(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	301
10.59.2.3 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)	302
10.59.2.4 fasp_solver_dstl_pcg(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	303
10.60pcg_mf.c File Reference	304
10.60.1 Detailed Description	304
10.60.2 Function Documentation	305
10.60.2.1 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
10.61pgcg.c File Reference	306
10.61.1 Detailed Description	306
10.61.2 Function Documentation	306
10.61.2.1 fasp_solver_dcsr_pgcg(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	306
10.62pgcg_mf.c File Reference	307
10.62.1 Detailed Description	307
10.62.2 Function Documentation	307
10.62.2.1 fasp_solver_pgcg(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	307
10.63pgcr.c File Reference	308
10.63.1 Detailed Description	308
10.63.2 Function Documentation	309
10.63.2.1 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)	309
10.63.2.2 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)	310
10.64pgmres.c File Reference	311
10.64.1 Detailed Description	311
10.64.2 Function Documentation	311
10.64.2.1 fasp_solver_bdcsl_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)	311
10.64.2.2 fasp_solver_dbsl_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)	312

10.64.2.3 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 S↵HORT prtvl)	313
10.64.2.4 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 S↵HORT prtvl)	313
10.65pgmres_mf.c File Reference	314
10.65.1 Detailed Description	314
10.65.2 Function Documentation	315
10.65.2.1 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 S↵HORT prtvl)	315
10.66pminres.c File Reference	315
10.66.1 Detailed Description	316
10.66.2 Function Documentation	317
10.66.2.1 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)	317
10.66.2.2 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)	317
10.66.2.3 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)	318
10.67pminres_mf.c File Reference	319
10.67.1 Detailed Description	319
10.67.2 Function Documentation	320
10.67.2.1 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)	320
10.68precond_bcsr.c File Reference	321
10.68.1 Detailed Description	321
10.68.2 Function Documentation	322
10.68.2.1 fasp_precond_block_diag_3(REAL *r, REAL *z, void *data)	322
10.68.2.2 fasp_precond_block_diag_3_amg(REAL *r, REAL *z, void *data)	323
10.68.2.3 fasp_precond_block_diag_4(REAL *r, REAL *z, void *data)	323
10.68.2.4 fasp_precond_block_lower_3(REAL *r, REAL *z, void *data)	324
10.68.2.5 fasp_precond_block_lower_3_amg(REAL *r, REAL *z, void *data)	325
10.68.2.6 fasp_precond_block_lower_4(REAL *r, REAL *z, void *data)	325
10.68.2.7 fasp_precond_block_SGS_3(REAL *r, REAL *z, void *data)	326
10.68.2.8 fasp_precond_block_SGS_3_amg(REAL *r, REAL *z, void *data)	327
10.68.2.9 fasp_precond_block_upper_3(REAL *r, REAL *z, void *data)	327
10.68.2.10 fasp_precond_block_upper_3_amg(REAL *r, REAL *z, void *data)	328
10.68.2.11 fasp_precond_sweeping(REAL *r, REAL *z, void *data)	329

10.69precond_bsr.c File Reference	329
10.69.1 Detailed Description	330
10.69.2 Function Documentation	330
10.69.2.1 fasp_precond_dbsr_amg(REAL *r, REAL *z, void *data)	330
10.69.2.2 fasp_precond_dbsr_amg_nk(REAL *r, REAL *z, void *data)	330
10.69.2.3 fasp_precond_dbsr_diag(REAL *r, REAL *z, void *data)	331
10.69.2.4 fasp_precond_dbsr_diag_nc2(REAL *r, REAL *z, void *data)	331
10.69.2.5 fasp_precond_dbsr_diag_nc3(REAL *r, REAL *z, void *data)	332
10.69.2.6 fasp_precond_dbsr_diag_nc5(REAL *r, REAL *z, void *data)	333
10.69.2.7 fasp_precond_dbsr_diag_nc7(REAL *r, REAL *z, void *data)	334
10.69.2.8 fasp_precond_dbsr_ilu(REAL *r, REAL *z, void *data)	335
10.69.2.9 fasp_precond_dbsr_nl_amli(REAL *r, REAL *z, void *data)	336
10.70precond_csr.c File Reference	336
10.70.1 Detailed Description	337
10.70.2 Function Documentation	337
10.70.2.1 fasp_precond_amg(REAL *r, REAL *z, void *data)	337
10.70.2.2 fasp_precond_amg_nk(REAL *r, REAL *z, void *data)	338
10.70.2.3 fasp_precond_amli(REAL *r, REAL *z, void *data)	338
10.70.2.4 fasp_precond_diag(REAL *r, REAL *z, void *data)	338
10.70.2.5 fasp_precond_famg(REAL *r, REAL *z, void *data)	339
10.70.2.6 fasp_precond_free(const SHORT precondition_type, precondition *pc)	339
10.70.2.7 fasp_precond_ilu(REAL *r, REAL *z, void *data)	340
10.70.2.8 fasp_precond_ilu_backward(REAL *r, REAL *z, void *data)	340
10.70.2.9 fasp_precond_ilu_forward(REAL *r, REAL *z, void *data)	340
10.70.2.10fasp_precond_nl_amli(REAL *r, REAL *z, void *data)	341
10.70.2.11fasp_precond_Schwarz(REAL *r, REAL *z, void *data)	341
10.70.2.12fasp_precond_setup(const SHORT precondition_type, AMG_param *amgparam, ILU_↵ param *iluparam, dCSRmat *A)	341
10.71precond_str.c File Reference	342
10.71.1 Detailed Description	343
10.71.2 Function Documentation	343
10.71.2.1 fasp_precond_dstr_blockgs(REAL *r, REAL *z, void *data)	343
10.71.2.2 fasp_precond_dstr_diag(REAL *r, REAL *z, void *data)	343
10.71.2.3 fasp_precond_dstr_ilu0(REAL *r, REAL *z, void *data)	343
10.71.2.4 fasp_precond_dstr_ilu0_backward(REAL *r, REAL *z, void *data)	344
10.71.2.5 fasp_precond_dstr_ilu0_forward(REAL *r, REAL *z, void *data)	344
10.71.2.6 fasp_precond_dstr_ilu1(REAL *r, REAL *z, void *data)	345

10.71.2.7 fasp_precond_dstr_ilu1_backward(REAL *r, REAL *z, void *data)	346
10.71.2.8 fasp_precond_dstr_ilu1_forward(REAL *r, REAL *z, void *data)	346
10.72pvfgmres.c File Reference	347
10.72.1 Detailed Description	347
10.72.2 Function Documentation	347
10.72.2.1 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 prtvl)	347
10.72.2.2 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 prtvl)	348
10.72.2.3 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)	349
10.73pvfgmres_mf.c File Reference	349
10.73.1 Detailed Description	350
10.73.2 Function Documentation	350
10.73.2.1 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)	350
10.74pvgmres.c File Reference	351
10.74.1 Detailed Description	351
10.74.2 Function Documentation	351
10.74.2.1 fasp_solver_bdcsr_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)	351
10.74.2.2 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 prtvl)	352
10.74.2.3 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)	353
10.74.2.4 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 prtvl)	354
10.75pvgmres_mf.c File Reference	355
10.75.1 Detailed Description	355
10.75.2 Function Documentation	355
10.75.2.1 fasp_solver_pvgmres(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)	355
10.76quadrature.c File Reference	356
10.76.1 Detailed Description	356

10.76.2 Function Documentation	356
10.76.2.1 fasp_gauss2d(const INT num_qp, const INT ncoor, REAL(*gauss)[3])	356
10.76.2.2 fasp_quad2d(const INT num_qp, const INT ncoor, REAL(*quad)[3])	357
10.77rap.c File Reference	357
10.77.1 Detailed Description	358
10.77.2 Function Documentation	358
10.77.2.1 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)	358
10.78schwarz_setup.c File Reference	358
10.78.1 Detailed Description	359
10.78.2 Function Documentation	359
10.78.2.1 fasp_dcsr_Schwarz_backward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)	359
10.78.2.2 fasp_dcsr_Schwarz_forward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)	359
10.78.2.3 fasp_Schwarz_get_block_matrix(Schwarz_data *Schwarz, INT nblk, INT *iblock, INT *jblock, INT *mask)	360
10.78.2.4 fasp_Schwarz_setup(Schwarz_data *Schwarz, Schwarz_param *param)	360
10.79smat.c File Reference	361
10.79.1 Detailed Description	361
10.79.2 Macro Definition Documentation	362
10.79.2.1 SWAP	362
10.79.3 Function Documentation	362
10.79.3.1 fasp_blas_smat_inv(REAL *a, const INT n)	362
10.79.3.2 fasp_blas_smat_inv_nc(REAL *a, const INT n)	362
10.79.3.3 fasp_blas_smat_inv_nc2(REAL *a)	362
10.79.3.4 fasp_blas_smat_inv_nc3(REAL *a)	363
10.79.3.5 fasp_blas_smat_inv_nc4(REAL *a)	363
10.79.3.6 fasp_blas_smat_inv_nc5(REAL *a)	363
10.79.3.7 fasp_blas_smat_inv_nc7(REAL *a)	364
10.79.3.8 fasp_blas_smat_invp_nc(REAL *a, const INT n)	364
10.79.3.9 fasp_blas_smat_Linfinity(REAL *A, const INT n)	365
10.79.3.10 fasp_iden_free(idenmat *A)	365
10.79.3.11 fasp_smat_identity(REAL *a, const INT n, const INT n2)	365
10.79.3.12 fasp_smat_identity_nc2(REAL *a)	366
10.79.3.13 fasp_smat_identity_nc3(REAL *a)	367
10.79.3.14 fasp_smat_identity_nc5(REAL *a)	367
10.79.3.15 fasp_smat_identity_nc7(REAL *a)	367

10.80 smoother_bsr.c File Reference	368
10.80.1 Detailed Description	369
10.80.2 Function Documentation	369
10.80.2.1 fasp_smoother_dbsr_gs(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark)	369
10.80.2.2 fasp_smoother_dbsr_gs1(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv)	369
10.80.2.3 fasp_smoother_dbsr_gs_ascend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	370
10.80.2.4 fasp_smoother_dbsr_gs_ascend1(dBSRmat *A, dvector *b, dvector *u)	370
10.80.2.5 fasp_smoother_dbsr_gs_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	371
10.80.2.6 fasp_smoother_dbsr_gs_descend1(dBSRmat *A, dvector *b, dvector *u)	371
10.80.2.7 fasp_smoother_dbsr_gs_order1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark)	372
10.80.2.8 fasp_smoother_dbsr_gs_order2(dBSRmat *A, dvector *b, dvector *u, INT *mark, REAL *work)	372
10.80.2.9 fasp_smoother_dbsr_ilu(dBSRmat *A, dvector *b, dvector *x, void *data)	373
10.80.2.10 fasp_smoother_dbsr_jacobi(dBSRmat *A, dvector *b, dvector *u)	373
10.80.2.11 fasp_smoother_dbsr_jacobi1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	374
10.80.2.12 fasp_smoother_dbsr_jacobi_setup(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	375
10.80.2.13 fasp_smoother_dbsr_sor(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL weight)	375
10.80.2.14 fasp_smoother_dbsr_sor1(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv, REAL weight)	376
10.80.2.15 fasp_smoother_dbsr_sor_ascend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)	376
10.80.2.16 fasp_smoother_dbsr_sor_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)	377
10.80.2.17 fasp_smoother_dbsr_sor_order(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight)	377
10.81 smoother_csr.c File Reference	378
10.81.1 Detailed Description	379
10.81.2 Function Documentation	379
10.81.2.1 fasp_smoother_dcsr_gs(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L)	379
10.81.2.2 fasp_smoother_dcsr_gs_cf(dvector *u, dCSRmat *A, dvector *b, INT L, INT *mark, const INT order)	379
10.81.2.3 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)	380
10.81.2.4 fasp_smoother_dcsr_ilu(dCSRmat *A, dvector *b, dvector *x, void *data)	380
10.81.2.5 fasp_smoother_dcsr_jacobi(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L)	381

10.81.2.6 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)	381
10.81.2.7 fasp_smoother_dcsr_L1diag(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L)	382
10.81.2.8 fasp_smoother_dcsr_sgs(dvector *u, dCSRmat *A, dvector *b, INT L)	382
10.81.2.9 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)	383
10.81.2.10 fasp_smoother_dcsr_sor_cf(dvector *u, dCSRmat *A, dvector *b, INT L, const REAL w, INT *mark, const INT order)	383
10.82 smoother_csr_cr.c File Reference	384
10.82.1 Detailed Description	384
10.82.2 Function Documentation	384
10.82.2.1 fasp_smoother_dcsr_gscr(INT pt, INT n, REAL *u, INT *ia, INT *ja, REAL *a, REAL *b, INT L, INT *CF)	384
10.83 smoother_csr_poly.c File Reference	385
10.83.1 Detailed Description	385
10.83.2 Function Documentation	386
10.83.2.1 fasp_smoother_dcsr_poly(dCSRmat *A, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L)	386
10.83.2.2 fasp_smoother_dcsr_poly_old(dCSRmat *A, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L)	386
10.84 smoother_str.c File Reference	386
10.84.1 Detailed Description	387
10.84.2 Function Documentation	388
10.84.2.1 fasp_generate_diaginv_block(dSTRmat *A, ivector *neigh, dvector *diaginv, ivector *pivot)	388
10.84.2.2 fasp_smoother_dstr_gs(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark)	388
10.84.2.3 fasp_smoother_dstr_gs1(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, REAL *diaginv)	388
10.84.2.4 fasp_smoother_dstr_gs_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	389
10.84.2.5 fasp_smoother_dstr_gs_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, const INT order)	389
10.84.2.6 fasp_smoother_dstr_gs_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	390
10.84.2.7 fasp_smoother_dstr_gs_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark)	390
10.84.2.8 fasp_smoother_dstr_jacobi(dSTRmat *A, dvector *b, dvector *u)	391
10.84.2.9 fasp_smoother_dstr_jacobi1(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	391
10.84.2.10 fasp_smoother_dstr_schwarz(dSTRmat *A, dvector *b, dvector *u, dvector *diaginv, ivector *pivot, ivector *neigh, ivector *order)	392

10.84.2.1	<code>fasp_smoother_dstr_sor(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, const REAL weight)</code>	392
10.84.2.2	<code>fasp_smoother_dstr_sor1(dSTRmat *A, dvector *b, dvector *u, const INT order, INT *mark, REAL *diaginv, const REAL weight)</code>	393
10.84.2.3	<code>fasp_smoother_dstr_sor_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)</code>	393
10.84.2.4	<code>fasp_smoother_dstr_sor_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, const INT order, const REAL weight)</code>	393
10.84.2.5	<code>fasp_smoother_dstr_sor_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)</code>	394
10.84.2.6	<code>fasp_smoother_dstr_sor_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight)</code>	394
10.85	<code>sparse_block.c</code> File Reference	395
10.85.1	Detailed Description	395
10.85.2	Function Documentation	396
10.85.2.1	<code>fasp_bdcsr_free(block_dCSRmat *A)</code>	396
10.85.2.2	<code>fasp_dbsr_getblk(dBSRmat *A, INT *Is, INT *Js, const INT m, const INT n, dBSRmat *B)</code>	397
10.85.2.3	<code>fasp_dbsr_getblk_dcsr(dBSRmat *A)</code>	397
10.85.2.4	<code>fasp_dbsr_Linfinity_dcsr(dBSRmat *A)</code>	398
10.85.2.5	<code>fasp_dcsr_getblk(dCSRmat *A, INT *Is, INT *Js, const INT m, const INT n, dCSRmat *B)</code>	398
10.86	<code>sparse_bsr.c</code> File Reference	399
10.86.1	Detailed Description	400
10.86.2	Function Documentation	400
10.86.2.1	<code>fasp_dbsr_alloc(const INT ROW, const INT COL, const INT NNZ, const INT nb, const INT storage_manner, dBSRmat *A)</code>	400
10.86.2.2	<code>fasp_dbsr_cp(dBSRmat *A, dBSRmat *B)</code>	400
10.86.2.3	<code>fasp_dbsr_create(const INT ROW, const INT COL, const INT NNZ, const INT nb, const INT storage_manner)</code>	401
10.86.2.4	<code>fasp_dbsr_diaginv(dBSRmat *A)</code>	401
10.86.2.5	<code>fasp_dbsr_diaginv2(dBSRmat *A, REAL *diaginv)</code>	402
10.86.2.6	<code>fasp_dbsr_diaginv3(dBSRmat *A, REAL *diaginv)</code>	402
10.86.2.7	<code>fasp_dbsr_diaginv4(dBSRmat *A, REAL *diaginv)</code>	403
10.86.2.8	<code>fasp_dbsr_diagLU(dBSRmat *A, REAL *DL, REAL *DU)</code>	403
10.86.2.9	<code>fasp_dbsr_diagLU2(dBSRmat *A, REAL *DL, REAL *DU)</code>	404
10.86.2.10	<code>fasp_dbsr_diagpref(dBSRmat *A)</code>	404
10.86.2.11	<code>fasp_dbsr_free(dBSRmat *A)</code>	405
10.86.2.12	<code>fasp_dbsr_getdiag(INT n, dBSRmat *A, REAL *diag)</code>	405
10.86.2.13	<code>fasp_dbsr_getdiaginv(dBSRmat *A)</code>	406

10.86.2.14	<code>asp_dbsr_null(dBSRmat *A)</code>	406
10.86.2.15	<code>asp_dbsr_trans(dBSRmat *A, dBSRmat *AT)</code>	407
10.87	<code>sparse_coo.c</code> File Reference	407
10.87.1	Detailed Description	407
10.87.2	Function Documentation	407
10.87.2.1	<code>asp_dcoo_alloc(const INT m, const INT n, const INT nnz, dCOOmat *A)</code>	407
10.87.2.2	<code>asp_dcoo_create(const INT m, const INT n, const INT nnz)</code>	408
10.87.2.3	<code>asp_dcoo_free(dCOOmat *A)</code>	408
10.87.2.4	<code>asp_dcoo_shift(dCOOmat *A, const INT offset)</code>	409
10.88	<code>sparse_csr.c</code> File Reference	409
10.88.1	Detailed Description	410
10.88.2	Function Documentation	410
10.88.2.1	<code>asp_dcsr_alloc(const INT m, const INT n, const INT nnz, dCSRmat *A)</code>	410
10.88.2.2	<code>asp_dcsr_compress(dCSRmat *A, dCSRmat *B, REAL dtol)</code>	411
10.88.2.3	<code>asp_dcsr_compress_inplace(dCSRmat *A, REAL dtol)</code>	411
10.88.2.4	<code>asp_dcsr_cp(dCSRmat *A, dCSRmat *B)</code>	412
10.88.2.5	<code>asp_dcsr_create(const INT m, const INT n, const INT nnz)</code>	412
10.88.2.6	<code>asp_dcsr_diagpref(dCSRmat *A)</code>	413
10.88.2.7	<code>asp_dcsr_free(dCSRmat *A)</code>	414
10.88.2.8	<code>asp_dcsr_getcol(const INT n, dCSRmat *A, REAL *col)</code>	414
10.88.2.9	<code>asp_dcsr_getdiag(INT n, dCSRmat *A, dvector *diag)</code>	415
10.88.2.10	<code>asp_dcsr_multicoloring(dCSRmat *A, INT *flags, INT *groups)</code>	415
10.88.2.11	<code>asp_dcsr_null(dCSRmat *A)</code>	416
10.88.2.12	<code>asp_dcsr_perm(dCSRmat *A, INT *P)</code>	416
10.88.2.13	<code>asp_dcsr_permz(dCSRmat *A, INT *p)</code>	416
10.88.2.14	<code>asp_dcsr_regdiag(dCSRmat *A, REAL value)</code>	417
10.88.2.15	<code>asp_dcsr_shift(dCSRmat *A, INT offset)</code>	417
10.88.2.16	<code>asp_dcsr_sort(dCSRmat *A)</code>	418
10.88.2.17	<code>asp_dcsr_sortz(dCSRmat *A, const SHORT isym)</code>	418
10.88.2.18	<code>asp_dcsr_symdiagscale(dCSRmat *A, dvector *diag)</code>	419
10.88.2.19	<code>asp_dcsr_sympat(dCSRmat *A)</code>	419
10.88.2.20	<code>asp_dcsr_trans(dCSRmat *A, dCSRmat *AT)</code>	419
10.88.2.21	<code>asp_dcsr_transz(dCSRmat *A, INT *p, dCSRmat *AT)</code>	420
10.88.2.22	<code>asp_icsr_cp(iCSRmat *A, iCSRmat *B)</code>	420
10.88.2.23	<code>asp_icsr_create(const INT m, const INT n, const INT nnz)</code>	421
10.88.2.24	<code>asp_icsr_free(iCSRmat *A)</code>	421
10.88.2.25	<code>asp_icsr_null(iCSRmat *A)</code>	422

10.88.2.2 fasp_icsr_trans(iCSRmat *A, iCSRmat *AT)	423
10.89 sparse_csrl.c File Reference	423
10.89.1 Detailed Description	424
10.89.2 Function Documentation	424
10.89.2.1 fasp_dcsrl_create(const INT num_rows, const INT num_cols, const INT num_nonzeros)	424
10.89.2.2 fasp_dcsrl_free(dCSRmat *A)	424
10.90 sparse_str.c File Reference	424
10.90.1 Detailed Description	425
10.90.2 Function Documentation	425
10.90.2.1 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)	425
10.90.2.2 fasp_dstr_cp(dSTRmat *A, dSTRmat *A1)	426
10.90.2.3 fasp_dstr_create(const INT nx, const INT ny, const INT nz, const INT nc, const INT nband, INT *offsets)	427
10.90.2.4 fasp_dstr_free(dSTRmat *A)	427
10.90.2.5 fasp_dstr_null(dSTRmat *A)	428
10.91 sparse_util.c File Reference	428
10.91.1 Detailed Description	429
10.91.2 Function Documentation	429
10.91.2.1 fasp_sparse_aat(INT *ia, INT *ja, REAL *a, INT *na, INT *ma, INT *iat, INT *jat, REAL *at)	429
10.91.2.2 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)	430
10.91.2.3 fasp_sparse_abybms(INT *ia, INT *ja, INT *ib, INT *jb, INT *nap, INT *map, INT *mbp, INT *ic, INT *jc)	430
10.91.2.4 fasp_sparse_aplbms(INT *ia, INT *ja, INT *ib, INT *jb, INT *nab, INT *mab, INT *ic, INT *jc)	431
10.91.2.5 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)	431
10.91.2.6 fasp_sparse_iit(INT *ia, INT *ja, INT *na, INT *ma, INT *iat, INT *jat)	431
10.91.2.7 fasp_sparse_MIS(dCSRmat *A)	432
10.91.2.8 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)	432
10.91.2.9 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)	433
10.91.2.10 fasp_sparse_wta(INT *jw, REAL *w, INT *ia, INT *ja, REAL *a, INT *nwp, INT *map, INT *jv, REAL *v, INT *nvp)	434
10.91.2.11 fasp_sparse_wtams(INT *jw, INT *ia, INT *ja, INT *nwp, INT *map, INT *jv, INT *nvp, INT *icp)	434
10.91.2.12 fasp_sparse_ytx(INT *jy, REAL *y, INT *jx, REAL *x, INT *nyp, INT *nxp, INT *icp, REAL *s)	434

10.91.2.1 fasp_sparse_ytxbig_(INT *jy, REAL *y, INT *nyp, REAL *x, REAL *s)	435
10.92spbcgs.c File Reference	435
10.92.1 Detailed Description	436
10.92.2 Function Documentation	437
10.92.2.1 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 prtvl)	437
10.92.2.2 fasp_solver_dbsr_spbcgs(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	437
10.92.2.3 fasp_solver_dcsr_spbcgs(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	438
10.92.2.4 fasp_solver_dstr_spbcgs(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	439
10.93spcg.c File Reference	440
10.93.1 Detailed Description	441
10.93.2 Function Documentation	442
10.93.2.1 fasp_solver_bdcsr_spcg(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	442
10.93.2.2 fasp_solver_dcsr_spcg(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	442
10.93.2.3 fasp_solver_dstr_spcg(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	443
10.94spgmres.c File Reference	444
10.94.1 Detailed Description	444
10.94.2 Function Documentation	444
10.94.2.1 fasp_solver_bdcsr_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)	444
10.94.2.2 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)	445
10.94.2.3 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)	446
10.94.2.4 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)	447
10.95spminres.c File Reference	448
10.95.1 Detailed Description	448
10.95.2 Function Documentation	449
10.95.2.1 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 prtvl)	449

10.95.2.2 fasp_solver_dcsr_spmminres(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	450
10.95.2.3 fasp_solver_dstr_spmminres(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	451
10.96spvgmres.c File Reference	452
10.96.1 Detailed Description	453
10.96.2 Function Documentation	453
10.96.2.1 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 prtvl)	453
10.96.2.2 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)	453
10.96.2.3 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)	454
10.96.2.4 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)	455
10.97threads.c File Reference	455
10.97.1 Detailed Description	456
10.97.2 Function Documentation	456
10.97.2.1 FASP_GET_START_END(INT procid, INT nprocs, INT n, INT *start, INT *end)	456
10.97.2.2 fasp_set_GS_threads(INT mythreads, INT its)	456
10.97.3 Variable Documentation	456
10.97.3.1 THDs_AMG_GS	456
10.97.3.2 THDs_CPR_gGS	457
10.97.3.3 THDs_CPR_IGS	457
10.98timing.c File Reference	457
10.98.1 Detailed Description	457
10.98.2 Function Documentation	457
10.98.2.1 fasp_gettime(REAL *time)	457
10.99vec.c File Reference	458
10.99.1 Detailed Description	458
10.99.2 Function Documentation	459
10.99.2.1 fasp_dvec_alloc(const INT m, dvector *u)	459
10.99.2.2 fasp_dvec_cp(dvector *x, dvector *y)	460
10.99.2.3 fasp_dvec_create(const INT m)	460
10.99.2.4 fasp_dvec_free(dvector *u)	461
10.99.2.5 fasp_dvec_isnan(dvector *u)	462

10.99.2.6 fasp_dvec_maxdiff(dvector *x, dvector *y)	. 462
10.99.2.7 fasp_dvec_null(dvector *x)	. 463
10.99.2.8 fasp_dvec_rand(const INT n, dvector *x)	. 463
10.99.2.9 fasp_dvec_set(INT n, dvector *x, REAL val)	. 464
10.99.2.10 fasp_dvec_symdiagscale(dvector *b, dvector *diag)	. 464
10.99.2.11 fasp_ivec_alloc(const INT m, ivec *u)	. 464
10.99.2.12 fasp_ivec_create(const INT m)	. 465
10.99.2.13 fasp_ivec_free(ivec *u)	. 465
10.99.2.14 fasp_ivec_set(const INT m, ivec *u)	. 466
10.100. Wrapper.c File Reference	. 466
10.100. Detailed Description	. 466
10.100. Function Documentation	. 467
10.100.2.1 fasp_fwrapper_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	. 467
10.100.2.2 fasp_fwrapper_krylov_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	. 467
10.100.2.3 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)	. 468
10.100.2.4 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)	. 468
Index	471

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.

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

The most updated version of FASP can be downloaded from

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

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

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

will give you the developer version of the FASP package.

Chapter 3

Building and Installation

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

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

```
$ make config
```

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

```
$ make install
```

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

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

```
$ wish fasp_install.tcl
```

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

```
$ make help
```

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

Chapter 4

Developers

Project leader:

- Xu, Jinchao (Penn State University, USA)

Project coordinator:

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

Current active developers (in alphabetic order):

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

With contributions from (in alphabetic order):

- Brannick, James (Penn State University, USA)
- Chen, Long (University of California, Irvine, USA)
- Huang, Feiteng (Sichuan University, China)
- Huang, Xuehai (Shanghai Jiaotong University, China)
- Qiao, Changhe (Penn State University, USA)
- Shu, Shi (Xiangtan University, China)
- Sun, Pengtao (University of Nevada, Las Vegas, USA)
- Yang, Kai (Penn State University, USA)
- Yue, Xiaoqiang (Xiangtan University, China)
- Wang, Lu (LLNL, USA)

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

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:

AMG_data	Data for AMG solvers	23
AMG_data_bsr	Data for multigrid levels. (BSR format)	24
AMG_param	Parameters for AMG solver	26
block_BSR	Block REAL matrix format for reservoir simulation	28
block_dCSRmat	Block REAL CSR matrix format	29
block_dvector	Block REAL vector structure	29
block_iCSRmat	Block INT CSR matrix format	30
block_ivector	Block INT vector structure	30
block_Reservoir	Block REAL matrix format for reservoir simulation	31
dBSRmat	Block sparse row storage matrix of REAL type	31
dCOOmat	Sparse matrix of REAL type in COO (or IJ) format	33
dCSRLmat	Sparse matrix of REAL type in CSRL format	33
dCSRmat	Sparse matrix of REAL type in CSR format	34
ddenmat	Dense matrix of REAL type	35
dSTRmat	Structure matrix of REAL type	35
dvector	Vector with n entries of REAL type	36
grid2d	Two dimensional grid data structure	37

iCOOmat	Sparse matrix of INT type in COO (or IJ) format	39
iCSRmat	Sparse matrix of INT type in CSR format	40
idenmat	Dense matrix of INT type	40
ILU_data	Data for ILU setup	41
ILU_param	Parameters for ILU	41
input_param	Input parameters	42
itsolver_param	Parameters passed to iterative solvers	49
ivector	Vector with n entries of INT type	51
Link	Struct for Links	51
linked_list	A linked list node	52
mallinfo	52
malloc_chunk	53
malloc_params	53
malloc_segment	54
malloc_state	54
malloc_tree_chunk	55
Mumps_data	Parameters for MUMPS interface	55
mxv_matfree	Matrix-vector multiplication, replace the actual matrix	56
nedmallinfo	56
Pardiso_data	Parameters for Intel MKL PARDISO interface	56
precond	Preconditioner data and action	57
precond_block_data	Data passed to the preconditioner for block preconditioning for block_dCSRmat format	57
precond_block_reservoir_data	Data passed to the preconditioner for reservoir simulation problems	59
precond_data	Data passed to the preconditioners	62
precond_data_bsr	Data passed to the preconditioners	64
precond_data_str	Data passed to the preconditioner for dSTRmat matrices	65
precond_diagbsr	Data passed to diagonal preconditioner for dBSRmat matrices	67
precond_diagstr	Data passed to diagonal preconditioner for dSTRmat matrices	67
precond_FASP_blkoi_data	Data passed to the preconditioner for preconditioning reservoir simulation problems	68
precond_sweeping_data	Data passed to the preconditioner for sweeping preconditioning	72

Schwarz_data	
Data for Schwarz methods	74
Schwarz_param	
Parameters for Schwarz method	75

Chapter 8

File Index

8.1 File List

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

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

coarsening_cr.c	
Coarsening with Brannick-Falgout strategy	149
coarsening_rs.c	
Coarsening with a modified Ruge-Stuben strategy	150
convert.c	
Some utilities for format conversion	152
dlmalloc.h	??
doxygen.h	
Main page for Doxygen documentation	156
eigen.c	
Subroutines for computing the extreme eigenvalues	156
famg.c	
Full AMG method as an iterative solver (main file)	157
fasp.h	
Main header file for FASP	158
fasp_block.h	
Header file for FASP block matrices	166
fasp_const.h	
Definition of all kinds of messages, including error messages, solver types, etc	169
fmgcycle.c	
Abstract non-recursive full multigrid cycle	189
formats.c	
Subroutines for matrix format conversion	189
givens.c	
Givens transformation	195
gmg_poisson.c	
GMG method as an iterative solver for Poisson Problem	196
graphics.c	
Subroutines for graphical output	203
hb_io.h	??
ilu_setup_bsr.c	
Setup incomplete LU decomposition for dBSRmat matrices	206
ilu_setup_csr.c	
Setup incomplete LU decomposition for dCSRmat matrices	207
ilu_setup_str.c	
Setup incomplete LU decomposition for dSTRmat matrices	208
init.c	
Initialize important data structures	209
input.c	
Read input parameters	214
interface_mumps.c	
Interface to MUMPS direct solvers	215
interface_pardiso.c	
Interface to Intel MKL PARDISO direct solvers	217
interface_samg.c	
Interface to SAMG solvers	218
interface_superlu.c	
Interface to SuperLU direct solvers	219
interface_umfpack.c	
Interface to UMFPACK direct solvers	220
interpolation.c	
Interpolation operators for AMG	221
interpolation_em.c	
Interpolation operators for AMG based on energy-min	223

io.c	Matrix/vector input/output subroutines	224
itsolver_bcsr.c	Iterative solvers for block_dCSRmat matrices	245
itsolver_bsr.c	Iterative solvers for dBSRmat matrices	249
itsolver_csr.c	Iterative solvers for dCSRmat matrices	253
itsolver_mf.c	Iterative solvers using matrix-free spmv operations	259
itsolver_str.c	Iterative solvers for dSTRmat matrices	262
lu.c	LU decomposition and direct solver for small dense matrices	266
malloc.c.h		??
memory.c	Memory allocation and deallocation subroutines	268
message.c	Output some useful messages	272
mgcycle.c	Abstract multigrid cycle – non-recursive version	275
mgrecur.c	Abstract multigrid cycle – recursive version	276
nedmalloc.h		??
ordering.c	Subroutines for ordering, merging, removing duplicated integers	277
parameters.c	Initialize, set, or print input data and parameters	283
pbcgs.c	Krylov subspace methods – Preconditioned BiCGstab	292
pbcgs_mf.c	Krylov subspace methods – Preconditioned BiCGstab (matrix free)	296
pcg.c	Krylov subspace methods – Preconditioned conjugate gradient	298
pcg_mf.c	Krylov subspace methods – Preconditioned conjugate gradient (matrix free)	304
pgcg.c	Krylov subspace methods – Preconditioned Generalized CG	306
pgcg_mf.c	Krylov subspace methods – Preconditioned Generalized CG (matrix free)	307
pgcr.c	Krylov subspace methods – Preconditioned GCR	308
pgmres.c	Krylov subspace methods – Right-preconditioned GMRes	311
pgmres_mf.c	Krylov subspace methods – Preconditioned GMRes (matrix free)	314
pminres.c	Krylov subspace methods – Preconditioned minimal residual	315
pminres_mf.c	Krylov subspace methods – Preconditioned minimal residual (matrix free)	319
precond_bcsr.c	Preconditioners for block_dCSRmat matrices	321
precond_bsr.c	Preconditioners for dBSRmat matrices	329

precond_csr.c	Preconditioners for dCSRmat matrices	336
precond_str.c	Preconditioners for dSTRmat matrices	342
pvfgmres.c	Krylov subspace methods – Preconditioned variable-restarting flexible GMRes	347
pvfgmres_mf.c	Krylov subspace methods – Preconditioned variable-restarting flexible GMRes (matrix free)	349
pvgmres.c	Krylov subspace methods – Preconditioned variable-restart GMRes	351
pvgmres_mf.c	Krylov subspace methods – Preconditioned variable-restarting GMRes (matrix free)	355
quadrature.c	Quadrature rules	356
rap.c	Tripple-matrix multiplication $R \cdot A \cdot P$	357
schwarz_setup.c	Setup phase for the Schwarz methods	358
smat.c	Simple operations for <i>small</i> dense matrices in row-major format	361
smoother_bsr.c	Smoothers for dBSRmat matrices	368
smoother_csr.c	Smoothers for dCSRmat matrices	378
smoother_csr_cr.c	Smoothers for dCSRmat matrices using compatible relaxation	384
smoother_csr_poly.c	Smoothers for dCSRmat matrices using poly. approx. to A^{-1}	385
smoother_str.c	Smoothers for dSTRmat matrices	386
sparse_block.c	Sparse matrix block operations	395
sparse_bsr.c	Sparse matrix operations for dBSRmat matrices	399
sparse_coo.c	Sparse matrix operations for dCOOmat matrices	407
sparse_csr.c	Sparse matrix operations for dCSRmat matrices	409
sparse_csrl.c	Sparse matrix operations for dCSRLmat matrices	423
sparse_str.c	Sparse matrix operations for dSTRmat matrices	424
sparse_util.c	Routines for sparse matrix operations	428
spbcgs.c	Krylov subspace methods – Preconditioned BiCGstab with safety net	435
spcg.c	Krylov subspace methods – Preconditioned conjugate gradient with safety net	440
spgmres.c	Krylov subspace methods – Preconditioned GMRes with safety net	444
spminres.c	Krylov subspace methods – Preconditioned minimal residual with safety net	448
spvgmres.c	Krylov subspace methods – Preconditioned variable-restart GMRes with safety net	452

threads.c	Get and set number of threads and assign work load for each thread	455
timing.c	Timing subroutines	457
vec.c	Simple operations for vectors	458
wrapper.c	Wrappers for accessing functions by advanced users	466

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
- [Pardiso_data pdata](#)
data for Intel MKL PARDISO
- [ivector cfmark](#)
pointer to the CF marker at level level_num
- [INT ILU_levels](#)
number of levels use ILU smoother
- [ILU_data LU](#)
ILU matrix for ILU smoother.
- [INT near_kernel_dim](#)

- *dimension of the near kernel for SAMG*
- [REAL ** near_kernel_basis](#)
basis of near kernel space for SAMG
- [INT Schwarz_levels](#)
number of levels use Schwarz smoother
- [Schwarz_data Schwarz](#)
data of Schwarz smoother
- [dvector w](#)
Temporary work space.
- [Mumps_data mumps](#)
data for MUMPS
- [INT cycle_type](#)
cycle type

9.1.1 Detailed Description

Data for AMG solvers.

Note

This is needed for the AMG solver/preconditioner.

Definition at line 722 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 diagin](#)
pointer to the diagonal inverse at level level_num
- [dCSRmat Ac](#)
pointer to the matrix at level level_num (csr format)
- `void *` [Numeric](#)
pointer to the numerical factorization from UMFPACK
- [Pardiso_data pdata](#)
data for Intel MKL PARDISO
- [dCSRmat PP](#)
pointer to the pressure block (only for reservoir simulation)
- `REAL *` [pw](#)
pointer to the auxiliary vectors for pressure block
- [dBSRmat SS](#)
pointer to the saturation block (only for reservoir simulation)
- `REAL *` [sw](#)
pointer to the auxiliary vectors for saturation block
- [dvector diaginv_SS](#)
pointer to the diagonal inverse of the saturation block at level level_num
- [ILU_data PP_LU](#)
ILU data for pressure block.
- [ivector cfmark](#)
pointer to the CF marker at level level_num
- `INT` [ILU_levels](#)
number of levels use ILU smoother
- [ILU_data LU](#)
ILU matrix for ILU smoother.
- `INT` [near_kernel_dim](#)
dimension of the near kernel for SAMG
- `REAL **` [near_kernel_basis](#)
basis of near kernel space for SAMG
- `dCSRmat *` [A_nk](#)
Matrix data for near kernel.
- `dCSRmat *` [P_nk](#)
Prolongation for near kernel.
- `dCSRmat *` [R_nk](#)
Restriction for near kernel.
- [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 presmoother
- [SHORT postsmooth_iter](#)
number of postsmoother
- [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_ifil](#)
level of fill-in for ILUs and ILUK
- [REAL ILU_droptol](#)
drop tolerance for ILU_t
- [REAL ILU_relax](#)
relaxation for ILUs

- [REAL ILU_permtol](#)
*permuted if $\text{permto} * |a(i,j)| > |a(i,i)|$*
- [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 583 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 209 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 265 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 148 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 108 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 304 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 342 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 * pdiri](#)
- [INT * ediri](#)
- [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 1127 of file fasp.h.

9.17.2 Field Documentation

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

Vertices of edges

Definition at line 1130 of file fasp.h.

9.17.2.2 INT edges

Number of edges

Definition at line 1141 of file fasp.h.

9.17.2.3 INT* ediri

Boundary flags (0 <=> interior edge)

Definition at line 1134 of file fasp.h.

9.17.2.4 INT* efather

Father edge or triangle

Definition at line 1137 of file fasp.h.

9.17.2.5 REAL(* p)[2]

Coordinates of vertices

Definition at line 1129 of file fasp.h.

9.17.2.6 INT* pdiri

Boundary flags (0 <=> interior point)

Definition at line 1133 of file fasp.h.

9.17.2.7 INT* pfather

Father point or edge

Definition at line 1136 of file fasp.h.

9.17.2.8 INT(* s)[3]

Edges of triangles

Definition at line 1132 of file fasp.h.

9.17.2.9 INT(* t)[3]

Vertices of triangles

Definition at line 1131 of file fasp.h.

9.17.2.10 INT* tfather

Father triangle

Definition at line 1138 of file fasp.h.

9.17.2.11 INT triangles

Number of triangles

Definition at line 1142 of file fasp.h.

9.17.2.12 INT vertices

Number of grid points

Definition at line 1140 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 239 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 178 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 127 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 * ijl](#)
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 400 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 ILUk
- [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 374 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 [infile](#) [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 1029 of file fasp.h.

9.23.2 Field Documentation

9.23.2.1 SHORT AMG_aggregation_type

aggregation type

Definition at line 1083 of file fasp.h.

9.23.2.2 INT AMG_aggressive_level

number of levels use aggressive coarsening

Definition at line 1088 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 1089 of file fasp.h.

9.23.2.4 SHORT AMG_amli_degree

degree of the polynomial used by AMLI cycle

Definition at line 1077 of file fasp.h.

9.23.2.5 INT AMG_coarse_dof

max number of coarsest level DOF

Definition at line 1071 of file fasp.h.

9.23.2.6 SHORT AMG_coarse_scaling

switch of scaling of the coarse grid correction

Definition at line 1076 of file fasp.h.

9.23.2.7 SHORT AMG_coarse_solver

coarse solver type

Definition at line 1075 of file fasp.h.

9.23.2.8 SHORT AMG_coarsening_type

coarsening type

Definition at line 1082 of file fasp.h.

9.23.2.9 SHORT AMG_cycle_type

type of cycle

Definition at line 1064 of file fasp.h.

9.23.2.10 SHORT AMG_ilu_levels

how many levels use ILU smoother

Definition at line 1074 of file fasp.h.

9.23.2.11 SHORT AMG_interpolation_type

interpolation type

Definition at line 1084 of file fasp.h.

9.23.2.12 SHORT AMG_levels

maximal number of levels

Definition at line 1063 of file fasp.h.

9.23.2.13 INT AMG_max_aggregation

max size of each aggregate

Definition at line 1095 of file fasp.h.

9.23.2.14 REAL AMG_max_row_sum

maximal row sum

Definition at line 1087 of file fasp.h.

9.23.2.15 INT AMG_maxit

number of iterations for AMG used as preconditioner

Definition at line 1073 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 1078 of file fasp.h.

9.23.2.17 INT AMG_pair_number

number of pairs in matching algorithm

Definition at line 1090 of file fasp.h.

9.23.2.18 SHORT AMG_polynomial_degree

degree of the polynomial smoother

Definition at line 1068 of file fasp.h.

9.23.2.19 SHORT AMG_postsmooth_iter

number of postsmoothing

Definition at line 1070 of file fasp.h.

9.23.2.20 SHORT AMG_presmooth_iter

number of presmoothing

Definition at line 1069 of file fasp.h.

9.23.2.21 REAL AMG_quality_bound

threshold for pair wise aggregation

Definition at line 1091 of file fasp.h.

9.23.2.22 REAL AMG_relaxation

over-relaxation parameter for SOR

Definition at line 1067 of file fasp.h.

9.23.2.23 INT AMG_Schwarz_levels

number of levels use Schwarz smoother

Definition at line 1079 of file fasp.h.

9.23.2.24 SHORT AMG_smooth_filter

use filter for smoothing the tentative prolongation or not

Definition at line 1097 of file fasp.h.

9.23.2.25 SHORT AMG_smooth_order

order for smoothers

Definition at line 1066 of file fasp.h.

9.23.2.26 SHORT AMG_smoother

type of smoother

Definition at line 1065 of file fasp.h.

9.23.2.27 REAL AMG_strong_coupled

strong coupled threshold for aggregate

Definition at line 1094 of file fasp.h.

9.23.2.28 REAL AMG_strong_threshold

strong threshold for coarsening

Definition at line 1085 of file fasp.h.

9.23.2.29 REAL AMG_tentative_smooth

relaxation factor for smoothing the tentative prolongation

Definition at line 1096 of file fasp.h.

9.23.2.30 REAL AMG_tol

tolerance for AMG if used as preconditioner

Definition at line 1072 of file fasp.h.

9.23.2.31 REAL AMG_truncation_threshold

truncation factor for interpolation

Definition at line 1086 of file fasp.h.

9.23.2.32 SHORT AMG_type

Type of AMG

Definition at line 1062 of file fasp.h.

9.23.2.33 REAL ILU_droptol

drop tolerance

Definition at line 1051 of file fasp.h.

9.23.2.34 INT ILU_ifil

level of fill-in

Definition at line 1050 of file fasp.h.

9.23.2.35 REAL ILU_permtol

permutation tolerance

Definition at line 1053 of file fasp.h.

9.23.2.36 REAL ILU_relax

scaling factor: add the sum of dropped entries to diagonal

Definition at line 1052 of file fasp.h.

9.23.2.37 SHORT ILU_type

ILU type for decomposition

Definition at line 1049 of file fasp.h.

9.23.2.38 char inifile[256]

ini file name

Definition at line 1036 of file fasp.h.

9.23.2.39 INT itsolver_maxit

maximal number of iterations for iterative solvers

Definition at line 1045 of file fasp.h.

9.23.2.40 REAL itsolver_tol

tolerance for iterative linear solver

Definition at line 1044 of file fasp.h.

9.23.2.41 SHORT output_type

type of output stream

Definition at line 1033 of file fasp.h.

9.23.2.42 SHORT precondition_type

type of preconditioner for iterative solvers

Definition at line 1042 of file fasp.h.

9.23.2.43 SHORT print_level

print level

Definition at line 1032 of file fasp.h.

9.23.2.44 INT problem_num

problem number to solve

Definition at line 1038 of file fasp.h.

9.23.2.45 INT restart

restart number used in GMRES

Definition at line 1046 of file fasp.h.

9.23.2.46 INT Schwarz_blksolver

type of Schwarz block solver

Definition at line 1059 of file fasp.h.

9.23.2.47 INT Schwarz_maxlvl

maximal levels

Definition at line 1057 of file fasp.h.

9.23.2.48 INT Schwarz_mmsize

maximal block size

Definition at line 1056 of file fasp.h.

9.23.2.49 INT Schwarz_type

type of Schwarz method

Definition at line 1058 of file fasp.h.

9.23.2.50 SHORT solver_type

type of iterative solvers

Definition at line 1041 of file fasp.h.

9.23.2.51 SHORT stop_type

type of stopping criteria for iterative solvers

Definition at line 1043 of file fasp.h.

9.23.2.52 char workdir[256]

working directory for data files

Definition at line 1037 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 1105 of file fasp.h.

9.24.2 Field Documentation

9.24.2.1 **SHORT** itsolver_type

solver type: see message.h

Definition at line 1107 of file fasp.h.

9.24.2.2 **INT** maxit

max number of iterations

Definition at line 1110 of file fasp.h.

9.24.2.3 **SHORT** precondition_type

preconditioner type: see message.h

Definition at line 1108 of file fasp.h.

9.24.2.4 **SHORT** print_level

print level: 0–10

Definition at line 1113 of file fasp.h.

9.24.2.5 **INT** restart

number of steps for restarting: for GMRES etc

Definition at line 1112 of file fasp.h.

9.24.2.6 **SHORT** stop_type

stopping criteria type

Definition at line 1109 of file fasp.h.

9.24.2.7 REAL tol

convergence tolerance

Definition at line 1111 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 356 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 1154 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 1171 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 **smbks**
- MALLINFO_FIELD_TYPE **hblks**
- MALLINFO_FIELD_TYPE **hblkhd**
- MALLINFO_FIELD_TYPE **usmbks**
- MALLINFO_FIELD_TYPE **fsmbks**
- 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 459 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 1013 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 **smbblks**
- size_t **hblks**
- size_t **hblkhd**
- size_t **usmbblks**
- size_t **fsmbblks**
- 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 Pardiso_data Struct Reference

Parameters for Intel MKL PARDISO interface.

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

Data Fields

- void * **pt** [64]

9.37.1 Detailed Description

Parameters for Intel MKL PARDISO interface.

Added on 11/28/2015

Definition at line 477 of file fasp.h.

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

- [fasp.h](#)

9.38 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

9.38.1 Detailed Description

Preconditioner data and action.

Note

This is the preconditioner structure for preconditioned iterative methods.

Definition at line 999 of file fasp.h.

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

- [fasp.h](#)

9.39 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](#)

9.39.1 Detailed Description

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

This is needed for the block preconditioner.

Definition at line 502 of file [fasp_block.h](#).

9.39.2 Field Documentation

9.39.2.1 [dCSRmat](#)* [A_diag](#)

data for each diagonal block

Definition at line 509 of file [fasp_block.h](#).

9.39.2.2 [block_dCSRmat](#)* [Abcsr](#)

problem data, the blocks

Definition at line 507 of file [fasp_block.h](#).

9.39.2.3 [AMG_param](#)* [amgparam](#)

parameters for AMG

Definition at line 521 of file [fasp_block.h](#).

9.39.2.4 [void](#)** [LU_diag](#)

LU decomposition for the diagonal blocks (for UMFPack)

Definition at line 517 of file [fasp_block.h](#).

9.39.2.5 [AMG_data](#)** [mgl](#)

AMG data for the diagonal blocks

Definition at line 520 of file [fasp_block.h](#).

9.39.2.6 dvector r

temp work space

Definition at line 511 of file fasp_block.h.

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

- [fasp_block.h](#)

9.40 precondition_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_lfil](#)
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 pressure-pressure 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 $(-I - Awr*Arr^{-1}*Arw)^{-1}$, Arr 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](#)

9.40.1 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 404 of file fasp_block.h.

9.40.2 Field Documentation

9.40.2.1 `precond_diagstr* diag`

the diagonal inverse for diagonal scaling

Definition at line 484 of file fasp_block.h.

9.40.2.2 dvector* diaginv

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

Definition at line 485 of file fasp_block.h.

9.40.2.3 dvector* diaginvS

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

Definition at line 487 of file fasp_block.h.

9.40.2.4 ivector* order

order for smoothing

Definition at line 489 of file fasp_block.h.

9.40.2.5 ivector* perf_idx

variable index for perf

Definition at line 477 of file fasp_block.h.

9.40.2.6 ivector* pivot

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

Definition at line 486 of file fasp_block.h.

9.40.2.7 ivector* pivotS

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

Definition at line 488 of file fasp_block.h.

9.40.2.8 dCSRmat* PP

pressure block after diagonal scaling

Definition at line 481 of file fasp_block.h.

9.40.2.9 dvector r

temporary dvector used to store and restore the residual

Definition at line 492 of file fasp_block.h.

9.40.2.10 dSTRmat* RR

Diagonal scaled reservoir block

Definition at line 479 of file fasp_block.h.

9.40.2.11 **SHORT** scaled

whether the matrix is scaled

Definition at line 476 of file fasp_block.h.

9.40.2.12 **dSTRmat*** SS

saturation block after diagonal scaling

Definition at line 482 of file fasp_block.h.

9.40.2.13 **REAL*** w

temporary work space for other usage

Definition at line 493 of file fasp_block.h.

9.40.2.14 **dCSRmat*** WW

Argumented well block

Definition at line 480 of file fasp_block.h.

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

- [fasp_block.h](#)

9.41 **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

9.41.1 Detailed Description

Data passed to the preconditioners.

Definition at line 795 of file fasp.h.

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

- [fasp.h](#)

9.42 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

9.42.1 Detailed Description

Data passed to the preconditioners.

Note

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

Definition at line 311 of file fasp_block.h.

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

- [fasp_block.h](#)

9.43 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

9.43.1 Detailed Description

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

Definition at line 891 of file fasp.h.

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

- [fasp.h](#)

9.44 precondition_diagbsr Struct Reference

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

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

Data Fields

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

9.44.1 Detailed Description

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

Note

This is needed for the diagonal preconditioner.

Definition at line 293 of file fasp_block.h.

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

- [fasp_block.h](#)

9.45 precondition_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

9.45.1 Detailed Description

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

Note

This is needed for the diagonal preconditioner.

Definition at line 983 of file fasp.h.

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

- [fasp.h](#)

9.46 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](#)

9.46.1 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 531 of file fasp_block.h.

9.46.2 Field Documentation

9.46.2.1 **block_BSR*** A

Part 1: Basic data.

whole jacobian system in block_BSRmat

Definition at line 536 of file fasp_block.h.

9.46.2.2 **dvector*** diaginv

inverse of the diagonal blocks of reservoir block

Definition at line 611 of file fasp_block.h.

9.46.2.3 **dvector*** diaginv_noscale

inverse of diagonal blocks for diagonal scaling

Definition at line 543 of file fasp_block.h.

9.46.2.4 **dvector*** diaginv_S

inverse of the diagonal blocks of saturation block

Definition at line 552 of file fasp_block.h.

9.46.2.5 **INT** maxit

max number of iterations

Definition at line 629 of file fasp_block.h.

9.46.2.6 **AMG_data*** mgl_data

AMG data for presure-presure block

Definition at line 557 of file fasp_block.h.

9.46.2.7 **ivector*** neigh

neighbor information of the reservoir block

Definition at line 547 of file fasp_block.h.

9.46.2.8 **ivector*** order

ordering of the reservoir block

Definition at line 548 of file fasp_block.h.

9.46.2.9 ivector* perf_idx

index of blocks which have perforation

Definition at line 618 of file fasp_block.h.

9.46.2.10 ivector* perf_neigh

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

Definition at line 619 of file fasp_block.h.

9.46.2.11 ivector* pivot

pivot for the GS smoothers for the reservoir matrix

Definition at line 612 of file fasp_block.h.

9.46.2.12 ivector* pivot_S

pivoting for the GS smoothers for saturation block

Definition at line 553 of file fasp_block.h.

9.46.2.13 dCSRmat* PP

pressure block

Definition at line 556 of file fasp_block.h.

9.46.2.14 dvector r

temporary dvector used to store and restore the residual

Definition at line 634 of file fasp_block.h.

9.46.2.15 INT restart

number of iterations for restart

Definition at line 630 of file fasp_block.h.

9.46.2.16 dBSRmat* RR

reservoir block

Definition at line 544 of file fasp_block.h.

9.46.2.17 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 542 of file fasp_block.h.

9.46.2.18 dBSRmat* SS

saturation block

Definition at line 551 of file fasp_block.h.

9.46.2.19 REAL tol

tolerance

Definition at line 631 of file fasp_block.h.

9.46.2.20 REAL* w

temporary work space for other usage

Definition at line 635 of file fasp_block.h.

9.46.2.21 dCSRmat* WW

Argumented well block

Definition at line 620 of file fasp_block.h.

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

- [fasp_block.h](#)

9.47 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](#)

9.47.1 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 648 of file fasp_block.h.

9.47.2 Field Documentation

9.47.2.1 block_dCSRmat* A

problem data, the sparse matrix

Definition at line 652 of file fasp_block.h.

9.47.2.2 block_dCSRmat* Ai

preconditioner data, the sparse matrix

Definition at line 653 of file fasp_block.h.

9.47.2.3 dCSRmat* local_A

local stiffness matrix for each layer

Definition at line 655 of file fasp_block.h.

9.47.2.4 ivector* local_index

local index for each layer

Definition at line 658 of file fasp_block.h.

9.47.2.5 void** local_LU

local LU decomposition (for UMFPack)

Definition at line 656 of file fasp_block.h.

9.47.2.6 INT NumLayers

number of layers

Definition at line 650 of file fasp_block.h.

9.47.2.7 dvector r

temporary dvector used to store and restore the residual

Definition at line 661 of file fasp_block.h.

9.47.2.8 REAL* w

temporary work space for other usage

Definition at line 662 of file fasp_block.h.

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

- [fasp_block.h](#)

9.48 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*

9.48.1 Detailed Description

Data for Schwarz methods.

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

Definition at line 505 of file fasp.h.

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

- [fasp.h](#)

9.49 Schwarz_param Struct Reference

Parameters for Schwarz method.

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

Data Fields

- [SHORT print_level](#)
 - print leve*
- [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*

9.49.1 Detailed Description

Parameters for Schwarz method.

Added on 05/14/2012

Definition at line 434 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 dCSRmat : 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 AMG_param : 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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data_bsr
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data_bsr
<i>param</i>	Pointer to AMG parameters: AMG_param

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_amli](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
AMLI – SOLVE phase.
- [INT fasp_amg_solve_nl_amli](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
Nonlinear AMLI – SOLVE phase.
- [void fasp_famg_solve](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
FMG – SOLVE phase.

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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

<code>lambda_max</code>	Maximal lambda
<code>lambda_min</code>	Minimal lambda
<code>degree</code>	Degree of polynomial approximation
<code>coef</code>	Coefficient of AMLI (output)

Author

Xiaozhe Hu

Date

01/23/2011

Definition at line 706 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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param
<i>level</i>	Current level

Author

Xiaozhe Hu

Date

01/23/2011

Note

AMLI polynomial computed by the best approximation of $1/x$. Refer to Johannes K. Kraus, Panayot S. Vassilevski, Ludmil T. Zikatanov, "Polynomial of best uniform approximation to x^{-1} and smoothing in two-level methods", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Zheng Li on 11/10/2014: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 45 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 AMG_data 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. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 269 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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param
<i>level</i>	Current level
<i>num_levels</i>	Total number of levels

Author

Xiaozhe Hu

Date

04/06/2010

Note

Nonlinear AMLI-cycle. Refer to Xiaozhe Hu, Panayot S. Vassilevski, Jinchao Xu "Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid", 2013.

Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 508 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 α
<i>A</i>	Pointer to block_dCSRmat 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 block_dCSRmat 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_mxv` (`dBSRmat *A`, `REAL *x`, `REAL *y`)
*Compute $y := A * x$.*
- void `fasp_blas_dbsr_mxv_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_mxm` (`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 dBSRmat 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 dBSRmat matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

Author

Xiaozhe Hu

Date

01/02/2014

Note

Works for general nb (Xiaozhe)

Definition at line 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 dBSRmat 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 dBSRmat matrix A
<i>B</i>	Pointer to the dBSRmat matrix B
<i>C</i>	Pointer to dBSRmat 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 dBSRmat 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_mnv_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 dBSRmat matrix
<i>x</i>	Pointer to the array x
<i>y</i>	Pointer to the array y

Author

Xiaozhe Hu

Date

01/02/2014

Note

Works for general nb (Xiaozhe)

Definition at line 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 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, 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_mxm (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 dCSRmat 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 dCSRmat matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

Author

Xiaozhe Hu

Date

02/22/2011

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

Definition at line 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 dCSRmat matrix
<i>alpha</i>	REAL factor alpha
<i>B</i>	Pointer to dCSRmat matrix
<i>beta</i>	REAL factor beta
<i>C</i>	Pointer to dCSRmat 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 dCSRmat 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 dCSRmat 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 dCSRmat matrix A
<i>B</i>	Pointer to the dCSRmat matrix B
<i>C</i>	Pointer to dCSRmat 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 dCSRmat 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 dCSRmat matrix R
<i>A</i>	Pointer to the dCSRmat matrix A
<i>P</i>	Pointer to the dCSRmat matrix P
<i>RAP</i>	Pointer to dCSRmat 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 dCSRmat matrix
<i>A</i>	pointer to the dCSRmat matrix
<i>P</i>	pointer to the dCSRmat matrix
<i>B</i>	pointer to dCSRmat 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 dCSRmat matrix R
<i>A</i>	Pointer to the dCSRmat matrix A
<i>P</i>	Pointer to the dCSRmat matrix P
<i>RAP</i>	Pointer to dCSRmat 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 dCSRmat matrix R
<i>A</i>	Pointer to the dCSRmat matrix A
<i>P</i>	Pointer to the dCSRmat matrix P
<i>B</i>	Pointer to dCSRmat 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 dCSRmat 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 dCSRmat 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 ReferenceBLAS2 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

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

Author

Xiaozhe Hu

Date

18/11/2011

Definition at line 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

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

Author

Xiaozhe Hu, Shiquan Zhang

Date

05/01/2010

Definition at line 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 $n \times n$ full matrix
<i>x</i>	Pointer to the REAL array with length <i>n</i>
<i>beta</i>	REAL factor beta
<i>y</i>	Pointer to the REAL array with length <i>n</i>
<i>n</i>	Length of array <i>x</i> and <i>y</i>

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_mnv (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 7×7 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 $y_s := y_s - Ass * x_s$, where 'A' is a 3*3 dense matrix, Ass is its saturaton 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 $y_s := y_s - Ass * x_s$, where 'A' is a 5*5 dense matrix, Ass is its saturaton 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 2×2 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

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 3×3 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

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 $y = \alpha A x + y$.

Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to dSTRmat 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 $y = Ax$.

Parameters

<i>A</i>	Pointer to dSTRmat 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

07/01/2009

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

07/01/2009

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_inf 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

A	Pointer to the matrix in dCSRmat 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

A	Pointer to the dCSRmat 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 dCSRmat 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 dCSRmat 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 iCSRmat 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 dCSRmat 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 dCSRmat : 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 AMG_param : 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 dCSRmat : 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 AMG_param : 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 dCSRmat 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 dCSRmat : 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 AMG_param : 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 [Pardiso_data](#)
Parameters for Intel MKL PARDISO interface.
- struct [Schwarz_data](#)
Data for Schwarz methods.
- struct [AMG_param](#)
Parameters for AMG solver.
- struct [AMG_data](#)
Data for AMG solvers.
- struct [precond_data](#)
Data passed to the preconditioners.
- struct [precond_data_str](#)
Data passed to the preconditioner for [dSTRmat](#) matrices.
- struct [precond_diagstr](#)
Data passed to diagonal preconditioner for [dSTRmat](#) matrices.
- struct [precond](#)
Preconditioner data and action.
- struct [mxv_matfree](#)
Matrix-vector multiplication, replace the actual matrix.
- struct [input_param](#)
Input parameters.
- struct [itsolver_param](#)
Parameters passed to iterative solvers.
- 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.

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

10.24.2 Macro Definition Documentation

10.24.2.1 `#define __FASP_HEADER__`

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

Definition at line 36 of file fasp.h.

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

absolute value of a

Definition at line 74 of file fasp.h.

10.24.2.3 `#define DIAGONAL_PREF OFF`

order each row such that diagonal appears first

Definition at line 58 of file fasp.h.

10.24.2.4 `#define DLMALLOC OFF`

use dlmalloc instead of standard malloc

Definition at line 47 of file fasp.h.

10.24.2.5 `#define FASP_GSRB 1`

MG level 0 use RedBlack Gauss Seidel Smoothing

Definition at line 1199 of file fasp.h.

10.24.2.6 `#define FASP_USE_ILU ON`

enable ILU or not

Definition at line 46 of file fasp.h.

10.24.2.7 #define FASP_VERSION 1.8

For external software package support.

faspsolver version

Definition at line 45 of file fasp.h.

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

is a >= b?

Definition at line 80 of file fasp.h.

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

Definition of >, >=, <, <=, and isnan.

is a > b?

Definition at line 79 of file fasp.h.

10.24.2.10 #define INT int

regular integer type: int or long

Definition at line 64 of file fasp.h.

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

is a == NAN?

Definition at line 83 of file fasp.h.

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

is a <= b?

Definition at line 82 of file fasp.h.

10.24.2.13 #define LONG long

long integer type

Definition at line 65 of file fasp.h.

10.24.2.14 #define LONGLONG long long

long integer type

Definition at line 66 of file fasp.h.

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

is $a < b$?

Definition at line 81 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 72 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 73 of file fasp.h.

10.24.2.18 **#define NEDMALLOC OFF**

use nedmalloc instead of standard malloc

Definition at line 48 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 88 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 89 of file fasp.h.

10.24.2.21 **#define REAL double**

float type

Definition at line 67 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 56 of file fasp.h.

10.24.2.23 #define SHORT short

FASP integer and floating point numbers.

short integer type

Definition at line 63 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 1194 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 1148 of file fasp.h.

10.24.3.15 typedef grid2d* pgrid2d

Grid in 2d

Definition at line 1146 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_ivec block_ivec`

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 SOLVER_PARDISO 34
- #define STOP_REL_RES 1

Definition of iterative solver stopping criteria types.

- #define STOP_REL_PRECRES 2
- #define STOP_MOD_REL_RES 3
- #define PREC_NULL 0

Definition of preconditioner type for iterative methods.

- #define PREC_DIAG 1
- #define PREC_AMG 2
- #define PREC_FMG 3

- #define [PREC_ILU](#) 4
- #define [PREC_SCHWARZ](#) 5
- #define [ILUk](#) 1
- Type of ILU methods.*
- #define [ILUt](#) 2
- #define [ILUtp](#) 3
- #define [SCHWARZ_FORWARD](#) 1
- Type of Schwarz smoother.*
- #define [SCHWARZ_BACKWARD](#) 2
- #define [SCHWARZ_SYMMETRIC](#) 3
- #define [CLASSIC_AMG](#) 1
- Definition of AMG types.*
- #define [SA_AMG](#) 2
- #define [UA_AMG](#) 3
- #define [PAIRWISE](#) 1
- Definition of aggregation types.*
- #define [VMB](#) 2
- #define [V_CYCLE](#) 1
- Definition of cycle types.*
- #define [W_CYCLE](#) 2
- #define [AMLI_CYCLE](#) 3
- #define [NL_AMLI_CYCLE](#) 4
- #define [SMOOTHER_JACOBI](#) 1
- Definition of standard smoother types.*
- #define [SMOOTHER_GS](#) 2
- #define [SMOOTHER_SGS](#) 3
- #define [SMOOTHER_CG](#) 4
- #define [SMOOTHER_SOR](#) 5
- #define [SMOOTHER_SSOR](#) 6
- #define [SMOOTHER_GSOR](#) 7
- #define [SMOOTHER_SGSOR](#) 8
- #define [SMOOTHER_POLY](#) 9
- #define [SMOOTHER_L1DIAG](#) 10
- #define [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 [G0PT](#) -5
- Type of vertices (DOFs) for coarsening.*
- #define [UNPT](#) -1
- #define [FGPT](#) 0
- #define [CGPT](#) 1
- #define [ISPT](#) 2

- #define `NO_ORDER` 0
Definition of smoothing order.
- #define `CF_ORDER` 1
- #define `USERDEFINED` 0
Type of ordering for smoothers.
- #define `CPFIRST` 1
- #define `FPFIRST` -1
- #define `ASCEND` 12
- #define `DESCEND` 21
- #define `BIGREAL` 1e+20
Some global constants.
- #define `SMALLREAL` 1e-20
- #define `SMALLREAL2` 1e-40
- #define `MAX_REFINE_LVL` 20
- #define `MAX_AMG_LVL` 20
- #define `MIN_CDOF` 20
- #define `MIN_CRATE` 0.9
- #define `MAX_CRATE` 20.0
- #define `MAX_RESTART` 20
- #define `MAX_STAG` 20
- #define `STAG_RATIO` 1e-4
- #define `OPENMP_HOLDS` 2000

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 177 of file fasp_const.h.

10.26.2.2 #define `ASCEND` 12

Ascending order

Definition at line 231 of file fasp_const.h.

10.26.2.3 `#define BIGREAL 1e+20`

Some global constants.

A large real number

Definition at line 237 of file fasp_const.h.

10.26.2.4 `#define CF_ORDER 1`

C/F order smoothing

Definition at line 223 of file fasp_const.h.

10.26.2.5 `#define CGPT 1`

Coarse grid points

Definition at line 216 of file fasp_const.h.

10.26.2.6 `#define CLASSIC_AMG 1`

Definition of AMG types.

classic AMG

Definition at line 162 of file fasp_const.h.

10.26.2.7 `#define COARSE_AC 4`

Aggressive coarsening

Definition at line 200 of file fasp_const.h.

10.26.2.8 `#define COARSE_CR 3`

Compatible relaxation

Definition at line 199 of file fasp_const.h.

10.26.2.9 `#define COARSE_MIS 5`

Aggressive coarsening based on MIS

Definition at line 201 of file fasp_const.h.

10.26.2.10 `#define COARSE_RS 1`

Definition of coarsening types.

Classical

Definition at line 197 of file fasp_const.h.

10.26.2.11 #define COARSE_RSP 2

Classical, with positive offdiags

Definition at line 198 of file fasp_const.h.

10.26.2.12 #define CPFIRST 1

C-points first order

Definition at line 229 of file fasp_const.h.

10.26.2.13 #define DESCEND 21

Descending order

Definition at line 232 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 215 of file fasp_const.h.

10.26.2.45 #define FPFIRST -1

F-points first order

Definition at line 230 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 213 of file fasp_const.h.

10.26.2.47 #define ILUk 1

Type of ILU methods.

ILUk

Definition at line 148 of file fasp_const.h.

10.26.2.48 #define ILUt 2

ILUt

Definition at line 149 of file fasp_const.h.

10.26.2.49 #define ILUtp 3

ILUtp

Definition at line 150 of file fasp_const.h.

10.26.2.50 #define INTERP_DIR 1

Definition of interpolation types.

Direct interpolation

Definition at line 206 of file fasp_const.h.

10.26.2.51 #define INTERP_ENG 3

energy minimization interpolation

Definition at line 208 of file fasp_const.h.

10.26.2.52 #define INTERP_STD 2

Standard interpolation

Definition at line 207 of file fasp_const.h.

10.26.2.53 #define ISPT 2

Isolated points

Definition at line 217 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 241 of file fasp_const.h.

10.26.2.63 `#define MAX_CRATE 20.0`

Maximal coarsening ratio

Definition at line 244 of file fasp_const.h.

10.26.2.64 `#define MAX_REFINE_LVL 20`

Maximal refinement level

Definition at line 240 of file fasp_const.h.

10.26.2.65 `#define MAX_RESTART 20`

Maximal restarting number

Definition at line 245 of file fasp_const.h.

10.26.2.66 `#define MAX_STAG 20`

Maximal number of stagnation times

Definition at line 246 of file fasp_const.h.

10.26.2.67 `#define MIN_CDOF 20`

Minimal number of coarsest variables

Definition at line 242 of file fasp_const.h.

10.26.2.68 `#define MIN_CRATE 0.9`

Minimal coarsening ratio

Definition at line 243 of file fasp_const.h.

10.26.2.69 `#define NL_AMLI_CYCLE 4`

Nonlinear AMLI-cycle

Definition at line 178 of file fasp_const.h.

10.26.2.70 `#define NO_ORDER 0`

Definition of smoothing order.

Natural order smoothing

Definition at line 222 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 248 of file fasp_const.h.

10.26.2.74 `#define PAIRWISE 1`

Definition of aggregation types.

pairwise aggregation

Definition at line 169 of file fasp_const.h.

10.26.2.75 `#define PREC_AMG 2`

with AMG precondition

Definition at line 140 of file fasp_const.h.

10.26.2.76 `#define PREC_DIAG 1`

with diagonal precondition

Definition at line 139 of file fasp_const.h.

10.26.2.77 `#define PREC_FMG 3`

with full AMG precondition

Definition at line 141 of file fasp_const.h.

10.26.2.78 `#define PREC_ILU 4`

with ILU precondition

Definition at line 142 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 138 of file fasp_const.h.

10.26.2.80 `#define PREC_SCHWARZ 5`

with Schwarz preconditioner

Definition at line 143 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 163 of file fasp_const.h.

10.26.2.88 #define SCHWARZ_BACKWARD 2

Backward ordering

Definition at line 156 of file fasp_const.h.

10.26.2.89 #define SCHWARZ_FORWARD 1

Type of Schwarz smoother.

Forward ordering

Definition at line 155 of file fasp_const.h.

10.26.2.90 #define SCHWARZ_SYMMETRIC 3

Symmetric smoother

Definition at line 157 of file fasp_const.h.

10.26.2.91 #define SMALLREAL 1e-20

A small real number

Definition at line 238 of file fasp_const.h.

10.26.2.92 #define SMALLREAL2 1e-40

An extremely small real number

Definition at line 239 of file fasp_const.h.

10.26.2.93 #define SMOOTHER_CG 4

CG as a smoother

Definition at line 186 of file fasp_const.h.

10.26.2.94 #define SMOOTHER_GS 2

Gauss-Seidel smoother

Definition at line 184 of file fasp_const.h.

10.26.2.95 #define SMOOTHER_GSOR 7

GS + SOR smoother

Definition at line 189 of file fasp_const.h.

10.26.2.96 #define SMOOTHER_JACOBI 1

Definition of standard smoother types.

Jacobi smoother

Definition at line 183 of file fasp_const.h.

10.26.2.97 #define SMOOTHER_L1DIAG 10

L1 norm diagonal scaling smoother

Definition at line 192 of file fasp_const.h.

10.26.2.98 #define SMOOTHER_POLY 9

Polynomial smoother

Definition at line 191 of file fasp_const.h.

10.26.2.99 #define SMOOTHER_SGS 3

Symmetric Gauss-Seidel smoother

Definition at line 185 of file fasp_const.h.

10.26.2.100 #define SMOOTHER_SGSOR 8

SGS + SSOR smoother

Definition at line 190 of file fasp_const.h.

10.26.2.101 #define SMOOTHER_SOR 5

SOR smoother

Definition at line 187 of file fasp_const.h.

10.26.2.102 #define SMOOTHER_SSOR 6

SSOR smoother

Definition at line 188 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`

Direct Solver: MUMPS

Definition at line 125 of file fasp_const.h.

10.26.2.113 `#define SOLVER_PARDISO 34`

Direct Solver: PARDISO

Definition at line 126 of file fasp_const.h.

10.26.2.114 `#define SOLVER_SBiCGstab 12`

BiCGstab with safety net

Definition at line 113 of file fasp_const.h.

10.26.2.115 `#define SOLVER_SCG 11`

Conjugate Gradient with safety net

Definition at line 112 of file fasp_const.h.

10.26.2.116 `#define SOLVER_SGCG 17`

GCG with safety net

Definition at line 118 of file fasp_const.h.

10.26.2.117 `#define SOLVER_SGMRES 14`

GMRes with safety net

Definition at line 115 of file fasp_const.h.

10.26.2.118 `#define SOLVER_SMinRes 13`

MinRes with safety net

Definition at line 114 of file fasp_const.h.

10.26.2.119 `#define SOLVER_SUPERLU 31`

Direct Solver: SuperLU

Definition at line 123 of file fasp_const.h.

10.26.2.120 #define SOLVER_SVFGMRES 16

Variable-restart FGMRES with safety net

Definition at line 117 of file fasp_const.h.

10.26.2.121 #define SOLVER_SVGMRES 15

Variable-restart GMRES with safety net

Definition at line 116 of file fasp_const.h.

10.26.2.122 #define SOLVER_UMFPACK 32

Direct Solver: UMFPack

Definition at line 124 of file fasp_const.h.

10.26.2.123 #define SOLVER_VFGMRES 6

Variable Restarting Flexible GMRES

Definition at line 108 of file fasp_const.h.

10.26.2.124 #define SOLVER_VGMRES 5

Variable Restarting GMRES

Definition at line 107 of file fasp_const.h.

10.26.2.125 #define STAG_RATIO 1e-4

Stagnation tolerance = tol*STAGRATIO

Definition at line 247 of file fasp_const.h.

10.26.2.126 #define STOP_MOD_REL_RES 3

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

Definition at line 133 of file fasp_const.h.

10.26.2.127 #define STOP_REL_PRECRES 2

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

Definition at line 132 of file fasp_const.h.

10.26.2.128 #define STOP_REL_RES 1

Definition of iterative solver stopping criteria types.

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

Definition at line 131 of file fasp_const.h.

10.26.2.129 **#define TRUE 1**

Definition of logic type.

logic TRUE

Definition at line 67 of file fasp_const.h.

10.26.2.130 **#define UA_AMG 3**

unsmoothed aggregation AMG

Definition at line 164 of file fasp_const.h.

10.26.2.131 **#define UNPT -1**

Undetermined points

Definition at line 214 of file fasp_const.h.

10.26.2.132 **#define USERDEFINED 0**

Type of ordering for smoothers.

User defined order

Definition at line 228 of file fasp_const.h.

10.26.2.133 **#define V_CYCLE 1**

Definition of cycle types.

V-cycle

Definition at line 175 of file fasp_const.h.

10.26.2.134 **#define VMB 2**

VMB aggregation

Definition at line 170 of file fasp_const.h.

10.26.2.135 **#define W_CYCLE 2**

W-cycle

Definition at line 176 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_fmgcycle](#) ([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_fmgcycle](#) ([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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

Author

Chensong Zhang

Date

02/27/2011

Modified by Chensong Zhang on 06/01/2012: fix a bug when there is only one level. Modified by Chensong Zhang on 02/27/2013: update direct solvers. Modified by Zheng Li on 11/10/2014: update direct solvers. Modified by Hongxuan Zhang on 12/15/2015: update direct solvers.

Definition at line 36 of file fmgcycle.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.
- [dCSRmat * fasp_format_dcsrl_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

<i>Ab</i>	Pointer to block_dCSRmat 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

<i>B</i>	Pointer to dBSRmat 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

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

Returns

dCSRmat matrix

Author

Zhiyang Zhou

Date

10/23/2010

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

Note

Works for general nb (Xiaozhe)

Definition at line 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 dCOOmat matrix
<i>B</i>	Pointer to dCSRmat matrix

Returns

FASP_SUCCESS if succeeded; otherwise, error information.

Author

Xuehai Huang

Date

08/10/2009

Definition at line 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 dCSRmat 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 dCSRmat matrix
<i>B</i>	Pointer to dCOOmat 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_dcsr ([dCSRmat](#) * *A*)

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

Parameters

<i>A</i>	Pointer to dCSRLmat 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 dSTRmat 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 dSTRmat matrix
<i>B</i>	Pointer to dCSRmat 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_functs.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 $y |beta * e_1 - H * y|$.

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 dBSRmat matrix
<i>filename</i>	File name

Author

Chunsheng Feng

Date

11/16/2013

Note

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

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

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

Definition at line 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 dBSRmat 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 dBSRmat 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 dCSRmat 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 dBSRmat matrix
<i>iludata</i>	Pointer to ILU_data
<i>iluparam</i>	Pointer to ILU_param

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 dCSRmat matrix
<i>iludata</i>	Pointer to ILU_data
<i>iluparam</i>	Pointer to ILU_param

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 dSTRmat
<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 oringinal structured matrix of REAL type
<i>LU</i>	Pointer to ILU structured matrix of REAL type

Author

Shiquan Zhang, Xiaozhe Hu

Date

11/08/2010

Note

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

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

Definition at line 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 AMG_data_bsr
------------	---

Author

Xiaozhe Hu

Date

2013/02/13

Definition at line 256 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 AMG_data
<i>param</i>	Pointer to AMG parameters

Author

Chensong Zhang

Date

2010/04/06

Modified by Chensong Zhang on 05/05/2013: Clean up param as well! Modified by Hongxuan Zhang on 12/15/2015: free internal memory for Intel MKL PARDISO.

Definition at line 185 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 ILU_data

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 ILU_data
----------------	-------------------------------------

Author

Chensong Zhang

Date

2010/04/03

Definition at line 300 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 ILU_data
----------------	-------------------------------------

Author

Chensong Zhang

Date

2010/03/23

Definition at line 321 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 337 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 AMG_data 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 dCSRmat 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 dCSRmat 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_pardiso.c File Reference

Interface to Intel MKL PARDISO direct solvers.

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

Functions

- [INT fasp_solver_pardiso](#) ([dCSRmat](#) *ptrA, [dvector](#) *b, [dvector](#) *u, const [SHORT](#) prtlvl)
Solve $Ax=b$ by PARDISO directly. Each row of A should be in ascending order w.r.t. column indices.

10.38.1 Detailed Description

Interface to Intel MKL PARDISO direct solvers.

Reference for Intel MKL PARDISO: <https://software.intel.com/en-us/node/470282>

10.38.2 Function Documentation

10.38.2.1 `int fasp_solver_pardiso (dCSRmat * ptrA, dvector * b, dvector * u, const SHORT prtlvl)`

Solve $Ax=b$ by PARDISO directly. Each row of A should be in ascending order w.r.t. column indices.

Parameters

<i>ptrA</i>	Pointer to a dCSRmat matrix
<i>b</i>	Pointer to the dvector of right-hand side term
<i>u</i>	Pointer to the dvector of solution
<i>prtlvl</i>	Output level

Author

Hongxuan Zhang

Date

11/28/2015

Definition at line 38 of file interface_pardiso.c.

10.39 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.

10.39.1 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)!

10.39.2 Function Documentation

10.39.2.1 [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 dCSRmat 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.

10.39.2.2 [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.

10.40 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.

10.40.1 Detailed Description

Interface to SuperLU direct solvers.

Reference for SuperLU: <http://crd-legacy.lbl.gov/~xiaoye/SuperLU/>

10.40.2 Function Documentation

10.40.2.1 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 dCSRmat 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.

10.41 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.

10.41.1 Detailed Description

Interface to UMFPACK direct solvers.

Reference for SuiteSparse: <http://faculty.cse.tamu.edu/davis/suitesparse.html>

10.41.2 Function Documentation

10.41.2.1 **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 dCSRmat 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.

10.42 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.

10.42.1 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.

10.42.2 Function Documentation

10.42.2.1 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 dCSRmat : 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.

10.42.2.2 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 dCSRmat : 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.

10.42.2.3 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 AMG_param : 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.

10.43 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.

10.43.1 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

10.43.2 Function Documentation

10.43.2.1 void [fasp_amg_interp_em](#) ([dCSRmat](#) * A, [ivector](#) * vertices, [dCSRmat](#) * P, [AMG_param](#) * param)

Energy-min interpolation.

Parameters

<i>A</i>	Pointer to dCSRmat : 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 dCSRmat matrix of resulted interpolation
<i>param</i>	Pointer to AMG_param : 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.

10.44 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

10.44.1 Detailed Description

Matrix/vector input/output subroutines.

Note

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

10.44.2 Function Documentation

10.44.2.1 void fasp_dbsr_print (dBSRmat * A)

Print out a `dBSRmat` matrix in coordinate format.

Parameters

A	Pointer to the <code>dBSRmat</code> 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.

10.44.2.2 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 dBSRmat 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.

10.44.2.3 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 dBSRmat 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.

10.44.2.4 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 dBSRmat 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.

10.44.2.5 `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.

10.44.2.6 `void fasp_dcoo_print (dCOOmat * A)`

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

Parameters

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

Author

Ziteng Wang

Date

12/24/2012

Definition at line 1423 of file io.c.

10.44.2.7 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.

10.44.2.8 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.

10.44.2.9 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 dCSRmat 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.

10.44.2.10 void fasp_dcsr_print (dCSRmat * *A*)

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

Parameters

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

Author

Xuehai Huang

Date

03/29/2009

Definition at line 1401 of file io.c.

10.44.2.11 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.

10.44.2.12 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 dCSRmat matrix A

Author

Chunsheng Feng

Date

11/14/2013

Definition at line 1531 of file io.c.

10.44.2.13 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`.

10.44.2.14 `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.

10.44.2.15 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.

10.44.2.16 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.

10.44.2.17 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.

10.44.2.18 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.

10.44.2.19 void fasp_dstr_print (**dSTRmat** * *A*)

Print out a **dSTRmat** matrix in coordinate format.

Parameters

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

Author

Ziteng Wang

Date

12/24/2012

Definition at line 1570 of file io.c.

10.44.2.20 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 dSTRmat

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.

10.44.2.21 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 dSTRmat 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.

10.44.2.22 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.

10.44.2.23 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.

10.44.2.24 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.

10.44.2.25 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.

10.44.2.26 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.

10.44.2.27 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.

10.44.2.28 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.

10.44.2.29 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.

10.44.2.30 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.

10.44.2.31 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.

10.44.2.32 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.

10.44.2.33 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.

10.44.2.34 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.

10.44.2.35 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.

10.44.2.36 `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.

10.44.3 Variable Documentation

10.44.3.1 INT dlength

Length of REAL in byte

Definition at line 14 of file io.c.

10.44.3.2 INT ilength

Length of INT in byte

Definition at line 13 of file io.c.

10.45 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.

10.45.1 Detailed Description

Iterative solvers for `block_dCSRmat` matrices.

10.45.2 Function Documentation

10.45.2.1 `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

<i>A</i>	Pointer to the coeff matrix in <code>block_dCSRmat</code> format
<i>b</i>	Pointer to the right hand side in dvector format
<i>x</i>	Pointer to the approx solution in dvector format
<i>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

Returns

Iteration number if converges; ERROR otherwise.

Author

Chensong Zhang

Date

11/25/2010

Definition at line 36 of file `itsolver_bcsr.c`.

10.45.2.2 `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 block_dCSRmat 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.

10.45.2.3 `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.

Parameters

<i>A</i>	Pointer to the coeff matrix in block_dCSRmat 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.

10.45.2.4 `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.

Parameters

<i>A</i>	Pointer to the coeff matrix in block_dCSRmat 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.

10.45.2.5 **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 block_dCSRmat 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 block_dCSRmat format
<i>local_A</i>	Pointer to the local coeff matrices in the dCSRmat 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.

10.46 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.

10.46.1 Detailed Description

Iterative solvers for [dBSRmat](#) matrices.

10.46.2 Function Documentation

10.46.2.1 [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 dBSRmat 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.

10.46.2.2 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 dBSRmat 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.

10.46.2.3 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 dBSRmat 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.

10.46.2.4 `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 dBSRmat 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 dBSRmat format
<i>P_nk</i>	Pointer to the prolongation for near kernel space in dBSRmat format
<i>R_nk</i>	Pointer to the restriction for near kernel space in dBSRmat format

Returns

Iteration number if converges; ERROR otherwise.

Author

Xiaozhe Hu

Date

05/26/2012

Definition at line 488 of file itsolver_bsr.c.

10.46.2.5 `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 dBSRmat 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.

10.46.2.6 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 dBSRmat 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.

10.46.2.7 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 dBSRmat 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.

10.47 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.

10.47.1 Detailed Description

Iterative solvers for `dCSRmat` matrices.

10.47.2 Function Documentation

10.47.2.1 `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

<code>A</code>	Pointer to the coeff matrix in <code>dCSRmat</code> format
<code>b</code>	Pointer to the right hand side in <code>dvector</code> format
<code>x</code>	Pointer to the approx solution in <code>dvector</code> 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

09/25/2009

Note

This is an abstract interface for iterative methods.

Definition at line 39 of file `itsolver_csr.c`.

10.47.2.2 `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 dCSRmat 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.

10.47.2.3 `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 dCSRmat 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.

10.47.2.4 `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 dCSRmat 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 dCSRmat format
<i>P_nk</i>	Pointer to the prolongation of near kernel space in dCSRmat format
<i>R_nk</i>	Pointer to the restriction of near kernel space in dCSRmat format

Returns

Iteration number if converges; ERROR otherwise.

Author

Xiaozhe Hu

Date

05/26/2014

Definition at line 611 of file itsolver_csr.c.

10.47.2.5 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 dCSRmat 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.

10.47.2.6 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 dCSRmat 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.

10.47.2.7 `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 dCSRmat 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 dCSRmat 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.

10.47.2.8 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 dCSRmat 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.

10.48 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.

10.48.1 Detailed Description

Iterative solvers using matrix-free spmv operations.

10.48.2 Function Documentation

10.48.2.1 `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 mxv_matfree 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.

10.48.2.2 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 mxv_matfree 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.

10.48.2.3 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 mxv_matfree 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.

10.49 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.

10.49.1 Detailed Description

Iterative solvers for [dSTRmat](#) matrices.

10.49.2 Function Documentation

10.49.2.1 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 dSTRmat 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.

10.49.2.2 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 dSTRmat 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.

10.49.2.3 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 dSTRmat 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.

10.49.2.4 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 dSTRmat 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.

10.49.2.5 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 dSTRmat 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.

10.50 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.

10.50.1 Detailed Description

LU decomposition and direct solver for small dense matrices.

10.50.2 Function Documentation**10.50.2.1 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.

10.50.2.2 **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.

10.51 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.

10.51.1 Detailed Description

Memory allocation and deallocation subroutines.

10.51.2 Function Documentation

10.51.2.1 void * fasp_mem_alloc (**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.

10.51.2.2 **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.

10.51.2.3 **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.

10.51.2.4 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.

10.51.2.5 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.

10.51.2.6 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.

10.51.2.7 void fasp_mem_usage ()

Show total allocated memory currently.

Author

Chensong Zhang

Date

2010/08/12

Definition at line 175 of file memory.c.

10.51.3 Variable Documentation**10.51.3.1 unsigned INT total_alloc_count = 0**

Total allocated memory amount.

total allocation times

Definition at line 35 of file memory.c.

10.51.3.2 unsigned INT total_alloc_mem = 0

total allocated memory

Definition at line 34 of file memory.c.

10.52 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.

10.52.1 Detailed Description

Output some useful messages.

Note

These routines are meant for internal use only.

10.52.2 Function Documentation

10.52.2.1 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.

10.52.2.2 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.

10.52.2.3 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.

10.52.2.4 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.

10.52.2.5 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.

10.52.2.6 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.

10.53 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.

10.53.1 Detailed Description

Abstract multigrid cycle – non-recursive version.

10.53.2 Function Documentation

10.53.2.1 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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param

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.

10.53.2.2 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: AMG_data_bsr
<i>param</i>	Pointer to AMG parameters: AMG_param

Author

Xiaozhe Hu

Date

08/07/2011

Definition at line 265 of file mgcycle.c.

10.54 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.

10.54.1 Detailed Description

Abstract multigrid cycle – recursive version.

Note

Not used any more. Will be removed! –Chensong

10.54.2 Function Documentation

10.54.2.1 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: AMG_data
<i>param</i>	Pointer to AMG parameters: AMG_param
<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.

10.55 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.

10.55.1 Detailed Description

Subroutines for ordering, merging, removing duplicated integers.

10.55.2 Function Documentation

10.55.2.1 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.

10.55.2.2 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.

10.55.2.3 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.

10.55.2.4 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.

10.55.2.5 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.

10.55.2.6 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.

10.55.2.7 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.

10.55.2.8 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.

10.55.2.9 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.

10.55.2.10 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.

10.56 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 `precond_data` with `AMG_param`.
- void `fasp_param_prec_to_amg` (`AMG_param` *amgparam, `precond_data` *pcdata)
Set `AMG_param` with `precond_data`.
- void `fasp_param_amg_to_prec_bsr` (`precond_data_bsr` *pcdata, `AMG_param` *amgparam)
Set `precond_data_bsr` with `AMG_param`.
- void `fasp_param_prec_to_amg_bsr` (`AMG_param` *amgparam, `precond_data_bsr` *pcdata)
Set `AMG_param` with `precond_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.

10.56.1 Detailed Description

Initialize, set, or print input data and parameters.

10.56.2 Function Documentation

10.56.2.1 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.

10.56.2.2 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.

10.56.2.3 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.

10.56.2.4 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.

10.56.2.5 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.

10.56.2.6 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.

10.56.2.7 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.

10.56.2.8 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.

10.56.2.9 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.

10.56.2.10 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.

10.56.2.11 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.

10.56.2.12 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.

10.56.2.13 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.

10.56.2.14 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.

10.56.2.15 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.

10.56.2.16 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.

10.56.2.17 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.

10.56.2.18 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.

10.56.2.19 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.

10.57 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 $Au=b$.

10.57.1 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

10.57.2 Function Documentation

10.57.2.1 INT fasp_solver_bdcsr_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.

10.57.2.2 `INT fasp_solver_dbsr_pbcgs (dBSRmat * A, dvector * b, dvector * u, precondition * pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl)`

Preconditioned BiCGstab method for solving $Au=b$.

Parameters

<i>A</i>	Pointer to 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.

10.57.2.3 `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.

10.57.2.4 `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.

10.58 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 $Au=b$.

10.58.1 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

10.58.2 Function Documentation

10.58.2.1 **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 mxv_matfree : 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.

10.59 pcg.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient.

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

Functions

- [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 $Au=b$.
- [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 $Au=b$.
- [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 $Au=b$.
- [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 $Au=b$.

10.59.1 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

10.59.2 Function Documentation

10.59.2.1 `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 block_dCSRmat : 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.

10.59.2.2 `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 dBSRmat : 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.

10.59.2.3 `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 dCSRmat : 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.

10.59.2.4 `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 dSTRmat : 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.

10.60 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 $Au=b$.

10.60.1 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

10.60.2 Function Documentation

10.60.2.1 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 mxv_matfree : 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.

10.61 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 genirilized conjugate gradient (GCG) method for solving $Au=b$.

10.61.1 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

10.61.2 Function Documentation

10.61.2.1 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 genirilized conjugate gradient (GCG) method for solving $Au=b$.

Parameters

<i>A</i>	Pointer to dCSRmat : 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 precond : 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.

10.62 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 $Au=b$.

10.62.1 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

10.62.2 Function Documentation

10.62.2.1 [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 mxv_matfree : 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.

10.63 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, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)
A preconditioned GCR method for solving $Au=b$.
- **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 $Au=b$.

10.63.1 Detailed Description

Krylov subspace methods – Preconditioned GCR.

10.63.2 Function Documentation

10.63.2.1 `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.

10.63.2.2 `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.

10.64 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 $Au=b$.

10.64.1 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

10.64.2 Function Documentation

- 10.64.2.1 [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 block_dCSRmat : 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.

10.64.2.2 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 dBSRmat : 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.

10.64.2.3 `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 dCSRmat : 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.

10.64.2.4 `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 dSTRmat : 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.

10.65 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.

10.65.1 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.

10.65.2 Function Documentation

10.65.2.1 **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 mxv_matfree : 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.

10.66 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 $Au=b$.

10.66.1 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

10.66.2 Function Documentation

10.66.2.1 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 block_dCSRmat : 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.

10.66.2.2 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 dCSRmat : 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.

10.66.2.3 `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 dSTRmat : 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.

10.67 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 $Au=b$.

10.67.1 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

10.67.2 Function Documentation

10.67.2.1 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 mxv_matfree : 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.

10.68 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

10.68.1 Detailed Description

Preconditioners for [block_dCSRmat](#) matrices.

10.68.2 Function Documentation

10.68.2.1 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.

10.68.2.2 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 110 of file precondition_bcsr.c.

10.68.2.3 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 175 of file precondition_bcsr.c.

10.68.2.4 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 271 of file precondition_bcsr.c.

10.68.2.5 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 353 of file precondition_bcsr.c.

10.68.2.6 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 427 of file precondition_bcsr.c.

10.68.2.7 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 688 of file precondition_bcsr.c.

10.68.2.8 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 806 of file precondition_bcsr.c.

10.68.2.9 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 525 of file precondition_bcsr.c.

10.68.2.10 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 607 of file precondition_bcsr.c.

10.68.2.11 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 916 of file precondition_bcsr.c.

10.69 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.

10.69.1 Detailed Description

Preconditioners for `dBSRmat` matrices.

10.69.2 Function Documentation

10.69.2.1 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`.

10.69.2.2 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.

10.69.2.3 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.

10.69.2.4 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`.

10.69.2.5 `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`.

10.69.2.6 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`.

10.69.2.7 `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`.

10.69.2.8 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`.

10.69.2.9 `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`.

10.70 `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

10.70.1 Detailed Description

Preconditioners for `dCSRmat` matrices.

10.70.2 Function Documentation

10.70.2.1 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.

10.70.2.2 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.

10.70.2.3 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.

10.70.2.4 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.

10.70.2.5 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.

10.70.2.6 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.

10.70.2.7 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.

10.70.2.8 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.

10.70.2.9 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.

10.70.2.10 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.

10.70.2.11 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.

10.70.2.12 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`.

10.71 `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)

10.71.1 Detailed Description

Preconditioners for [dSTRmat](#) matrices.

10.71.2 Function Documentation

10.71.2.1 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.

10.71.2.2 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.

10.71.2.3 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.

10.71.2.4 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.

10.71.2.5 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.

10.71.2.6 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.

10.71.2.7 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.

10.71.2.8 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.

10.72 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.

10.72.1 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](#)

10.72.2 Function Documentation

10.72.2.1 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.

10.72.2.2 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 dCSRmat : 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.

10.72.2.3 `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 dCSRmat : 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.

10.73 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.

10.73.1 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](#)

10.73.2 Function Documentation

10.73.2.1 `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 mxv_matfree : 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.

10.74 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.

10.74.1 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

10.74.2 Function Documentation

10.74.2.1 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 dCSRmat : 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

Chensong Zhang

Date

04/05/2013

Definition at line 393 of file pvgmres.c.

10.74.2.2 `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 dCSRmat : 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.

10.74.2.3 `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 dCSRmat : 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.

10.74.2.4 `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 dCSRmat : 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.

10.75 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.

10.75.1 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.

10.75.2 Function Documentation

10.75.2.1 [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 mxv_matfree : 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.

10.76 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.

10.76.1 Detailed Description

Quadrature rules.

10.76.2 Function Documentation

10.76.2.1 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`.

10.76.2.2 `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`.

10.77 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 $R \cdot A \cdot P$.

10.77.1 Detailed Description

Tripple-matrix multiplication $R \cdot A \cdot P$.

C-version by Ludmil Zikatanov 2010-04-08

tested 2010-04-08

10.77.2 Function Documentation

10.77.2.1 `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.

10.78 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.

10.78.1 Detailed Description

Setup phase for the Schwarz methods.

10.78.2 Function Documentation

10.78.2.1 `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.

10.78.2.2 `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.

10.78.2.3 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.

10.78.2.4 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.

10.79 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.*

10.79.1 Detailed Description

Simple operations for *small* dense matrices in row-major format.

10.79.2 Macro Definition Documentation

10.79.2.1 `#define SWAP(a, b) {temp=(a);(a)=(b);(b)=temp;}`

swap two numbers

Definition at line 9 of file smat.c.

10.79.3 Function Documentation

10.79.3.1 `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.

10.79.3.2 `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.

10.79.3.3 `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.

10.79.3.4 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.

10.79.3.5 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.

10.79.3.6 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.

10.79.3.7 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.

10.79.3.8 void fasp_blas_smat_inv_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.

10.79.3.9 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.

10.79.3.10 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.

10.79.3.11 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.

10.79.3.12 void fasp_smat_identity_nc2 (REAL * a)

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

Parameters

a	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.

10.79.3.13 void fasp_smat_identity_nc3 (REAL * a)

Set a 3*3 full matrix to be a identity.

Parameters

a	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.

10.79.3.14 void fasp_smat_identity_nc5 (REAL * a)

Set a 5*5 full matrix to be a identity.

Parameters

a	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.

10.79.3.15 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.

10.80 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.

10.80.1 Detailed Description

Smoothers for `dBSRmat` matrices.

10.80.2 Function Documentation

10.80.2.1 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`.

10.80.2.2 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 dBSRmat : 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.

10.80.2.3 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 dBSRmat : 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.

10.80.2.4 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 dBSRmat : 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.

10.80.2.5 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 dBSRmat : 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.

10.80.2.6 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 dBSRmat : 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.

10.80.2.7 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 dBSRmat : 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.

10.80.2.8 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 dBSRmat : 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.

10.80.2.9 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 dBSRmat : 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.

10.80.2.10 void fasp_smoother_dbsr_jacobi (dBSRmat * *A*, dvector * *b*, dvector * *u*)

Jacobi relaxation.

Parameters

<i>A</i>	Pointer to dBSRmat : 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.

10.80.2.11 void fasp_smoother_dbsr_jacobi1 (dBSRmat * *A*, dvector * *b*, dvector * *u*, REAL * *diaginv*)

Jacobi relaxation.

Parameters

<i>A</i>	Pointer to dBSRmat : 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.

10.80.2.12 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 dBSRmat : 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.

10.80.2.13 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 dBSRmat : 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.

10.80.2.14 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 dBSRmat : 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.

10.80.2.15 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 dBSRmat : 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.

```
10.80.2.16 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 dBSRmat : 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.

```
10.80.2.17 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 dBSRmat : 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.

10.81 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.

10.81.1 Detailed Description

Smoothers for `dCSRmat` matrices.

10.81.2 Function Documentation

10.81.2.1 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 <code>dvector</code> : 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 <code>dvector</code> : 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`.

10.81.2.2 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 dBSRmat : 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.

10.81.2.3 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.

10.81.2.4 void fasp_smoother_dcsr_ilu (dCSRmat * *A*, dvector * *b*, dvector * *x*, void * *data*)

ILU method as a smoother.

Parameters

<i>A</i>	Pointer to dBSRmat : 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.

10.81.2.5 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 dBSRmat : 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.

10.81.2.6 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 dBSRmat : the coefficient matrix
b	Pointer to dvector: the right hand side
L	Number of iterations
w	Over-relaxation weight

Author

Xiaozhe Hu

Date

2010/11/12

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

Definition at line 1145 of file smoother_csr.c.

10.81.2.7 `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 dBSRmat : 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.

10.81.2.8 `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 dBSRmat : 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.

```
10.81.2.9 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 dBSRmat : 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.

```
10.81.2.10 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 dBSRmat : 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.

10.82 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.

10.82.1 Detailed Description

Smoothers for [dCSRmat](#) matrices using compatible relaxation.

Note

Restricted-smoothers for compatible relaxation, C/F smoothing, etc.

10.82.2 Function Documentation

10.82.2.1 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.

10.83 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<Z2010

10.83.1 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.

10.83.2 Function Documentation

10.83.2.1 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.

10.83.2.2 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<Z2010

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.

10.84 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.

10.84.1 Detailed Description

Smoothers for `dSTRmat` matrices.

10.84.2 Function Documentation

10.84.2.1 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 dCSRmat : 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.

10.84.2.2 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 dCSRmat : 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.

10.84.2.3 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 dCSRmat : 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.

10.84.2.4 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 dCSRmat : 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.

10.84.2.5 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 dCSRmat : 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.

10.84.2.6 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 dCSRmat : 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.

10.84.2.7 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 dCSRmat : 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.

10.84.2.8 void fasp_smoother_dstr_jacobi (dSTRmat * *A*, dvector * *b*, dvector * *u*)

Jacobi method as the smoother.

Parameters

<i>A</i>	Pointer to dCSRmat : 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.

10.84.2.9 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 dCSRmat : 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.

10.84.2.10 void fasp_smoother_dstr_schwarz (dSTRmat * *A*, dvector * *b*, dvector * *u*, dvector * *diagin*, iverector * *pivot*, iverector * *neigh*, iverector * *order*)

Schwarz method as the smoother.

Parameters

<i>A</i>	Pointer to dCSRmat : 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 iverector: the pivot of diagonal blocks
<i>neigh</i>	Pointer to iverector: neighborhoods
<i>order</i>	Pointer to iverector: the smoothing order

Author

Xiaozhe Hu

Date

10/01/2011

Definition at line 1643 of file smoother_str.c.

10.84.2.11 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 dCSRmat : 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.

10.84.2.12 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 dCSRmat : 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.

10.84.2.13 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 dCSRmat : 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.

10.84.2.14 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 dCSRmat : 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.

10.84.2.15 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 dCSRmat : 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.

10.84.2.16 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 dCSRmat : 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.

10.85 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

10.85.1 Detailed Description

Sparse matrix block operations.

10.85.2 Function Documentation

10.85.2.1 void fasp_bdcsr_free (block_dCSRmat * A)

Free block CSR sparse matrix data memory space.

Parameters

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

Author

Xiaozhe Hu

Date

04/18/2014

Definition at line 30 of file sparse_block.c.

10.85.2.2 **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 dBSRmat BSR matrix
<i>B</i>	Pointer to dBSRmat 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.

10.85.2.3 **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.

10.85.2.4 **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.

10.85.2.5 **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 dCSRmat matrix
<i>B</i>	Pointer to dCSRmat 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.

10.86 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 $DU = \text{diag}(U^{-1})$.

10.86.1 Detailed Description

Sparse matrix operations for `dBSRmat` matrices.

10.86.2 Function Documentation

10.86.2.1 `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

<i>ROW</i>	Number of rows of block
<i>COL</i>	Number of columns of block
<i>NNZ</i>	Number of nonzero blocks
<i>nb</i>	Dimension of each block
<i>storage_manner</i>	Storage manner for each sub-block
<i>A</i>	Pointer to new <code>dBSRmat</code> matrix

Author

Xiaozhe Hu

Date

10/26/2010

Definition at line 87 of file `sparse_bsr.c`.

10.86.2.2 `void fasp_dbsr_cp (dBSRmat * A, dBSRmat * B)`

copy a `dCSRmat` to a new one B=A

Parameters

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

Author

Xiaozhe Hu

Date

08/07/2011

Definition at line 181 of file sparse_bsr.c.

10.86.2.3 [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.

10.86.2.4 [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 dBSRmat 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.

10.86.2.5 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 dBSRmat 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.

10.86.2.6 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 dBSRmat 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.

10.86.2.7 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 dBSRmat 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.

10.86.2.8 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 dBSRmat matrix
----------	---

<i>DL</i>	Pointer to the $\text{diag}(L^{-1})$
<i>DU</i>	Pointer to the $\text{diag}(U^{-1})$

Returns

BSR matrix after scaling

Author

Xiaozhe Hu

Date

04/02/2014

Definition at line 1449 of file sparse_bsr.c.

10.86.2.9 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 dBSRmat matrix
<i>DL</i>	Pointer to the $\text{diag}(L^{-1})$
<i>DU</i>	Pointer to the $\text{diag}(U^{-1})$

Returns

BSR matrix after scaling

Author

Zheng Li, Xiaozhe Hu

Date

06/17/2014

Definition at line 1677 of file sparse_bsr.c.

10.86.2.10 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.

10.86.2.11 void fasp_dbsr_free (dBSRmat * *A*)

Free memory space for BSR format sparse matrix.

Parameters

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

Author

Xiaozhe Hu

Date

10/26/2010

Definition at line 133 of file sparse_bsr.c.

10.86.2.12 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.

10.86.2.13 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.

10.86.2.14 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.

10.86.2.15 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.

10.87 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.

10.87.1 Detailed Description

Sparse matrix operations for dCOOmat matrices.

10.87.2 Function Documentation

10.87.2.1 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 dCSRmat matrix

Author

Xiaozhe Hu

Date

03/25/2013

Definition at line 62 of file sparse_coo.c.

10.87.2.2 [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.

10.87.2.3 void fasp_dcoo_free ([dCOOmat](#) * A)

Free IJ sparse matrix data memory space.

Parameters

A	Pointer to the dCOOmat matrix
-----	---

Author

Chensong Zhang

Date

2010/04/03

Definition at line 94 of file sparse_coo.c.

10.87.2.4 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.

10.88 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 $*tnz$)
- 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.

10.88.1 Detailed Description

Sparse matrix operations for dCSRmat matrices.

10.88.2 Function Documentation

10.88.2.1 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 dCSRmat matrix

Author

Chensong Zhang

Date

2010/04/06

Definition at line 125 of file sparse_csr.c.

10.88.2.2 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 dCSRmat CSR matrix
<i>B</i>	Pointer to dCSRmat 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.

10.88.2.3 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 dCSRmat 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.

10.88.2.4 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.

10.88.2.5 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.

10.88.2.6 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.

10.88.2.7 `void fasp_dcsr_free (dCSRmat * A)`

Free CSR sparse matrix data memory space.

Parameters

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

Author

Chensong Zhang

Date

2010/04/06

Definition at line 166 of file sparse_csr.c.

10.88.2.8 `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 dCSRmat 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.

10.88.2.9 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 dCSRmat 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.

10.88.2.10 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 dCSRmat
<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.

10.88.2.11 void fasp_dcsr_null (dCSRmat * A)

Initialize CSR sparse matrix.

Parameters

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

Author

Chensong Zhang

Date

2010/04/03

Definition at line 204 of file sparse_csr.c.

10.88.2.12 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 dCSRmat 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.

10.88.2.13 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 dCSRmat 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`.

10.88.2.14 `SHORT fasp_dcsr_regdiag (dCSRmat * A, REAL value)`

Regularize diagonal entries of a CSR sparse matrix.

Parameters

A	Pointer to the dCSRmat 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`.

10.88.2.15 `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.

10.88.2.16 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 dCSRmat matrix
----------	---

Author

Shiquan Zhang

Date

06/10/2010

Definition at line 358 of file sparse_csr.c.

10.88.2.17 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 dCSRmat 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.

10.88.2.18 void fasp_dcsr_symdiagscale (dCSRmat * A, dvector * diag)

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

Parameters

A	Pointer to the dCSRmat matrix
diag	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.

10.88.2.19 dCSRmat fasp_dcsr_sympat (dCSRmat * A)

Get symmetric part of a dCSRmat matrix.

Parameters

*A	pointer to the dCSRmat matrix
----	-------------------------------

Returns

symmetrized the dCSRmat matrix

Author

Xiaozhe Hu

Date

03/21/2011

Definition at line 1232 of file sparse_csr.c.

10.88.2.20 void fasp_dcsr_trans (dCSRmat * A, dCSRmat * AT)

Find transpose of dCSRmat matrix A.

Parameters

A	Pointer to the dCSRmat matrix
---	-------------------------------

<i>AT</i>	Pointer to the transpose of dCSRmat 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.

10.88.2.21 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 dCSRmat 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.

10.88.2.22 void fasp_icsr_cp (iCSRmat * A, iCSRmat * B)

Copy a [iCSRmat](#) to a new one B=A.

Parameters

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

Author

Chensong Zhang

Date

05/16/2013

Definition at line 698 of file sparse_csr.c.

10.88.2.23 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.

10.88.2.24 void fasp_icshr_free (iCSRmat * *A*)

Free CSR sparse matrix data memory space.

Parameters

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

Author

Chensong Zhang

Date

2010/04/06

Definition at line 185 of file sparse_csr.c.

10.88.2.25 void fasp_icsr_null (iCSRmat * A)

Initialize CSR sparse matrix.

Parameters

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

Author

Chensong Zhang

Date

2010/04/03

Definition at line 221 of file sparse_csr.c.

10.88.2.26 void fasp_icsr_trans ([iCSRmat](#) * *A*, [iCSRmat](#) * *AT*)

Find transpose of [iCSRmat](#) matrix *A*.

Parameters

<i>A</i>	Pointer to the iCSRmat matrix <i>A</i>
<i>AT</i>	Pointer to the iCSRmat 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.

10.89 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.

10.89.1 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.

10.89.2 Function Documentation

10.89.2.1 [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.

10.89.2.2 void fasp_dcsrl_free ([dCSRLmat](#) * *A*)

Destroy a [dCSRLmat](#) object.

Parameters

<i>A</i>	Pointer to the dCSRLmat type matrix
----------	---

Author

Zhiyang Zhou

Date

01/07/2011

Definition at line 58 of file sparse_csrl.c.

10.90 [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.

10.90.1 Detailed Description

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

10.90.2 Function Documentation

10.90.2.1 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 dSTRmat matrix

Author

Shiquan Zhang, Xiaozhe Hu

Date

05/17/2010

Definition at line 109 of file sparse_str.c.

10.90.2.2 void fasp_dstr_cp (dSTRmat * A, dSTRmat * A1)

Copy a [dSTRmat](#) to a new one A1=A.

Parameters

<i>A</i>	Pointer to the dSTRmat matrix
<i>A1</i>	Pointer to the dSTRmat matrix

Author

Zhiyang Zhou

Date

04/21/2010

Definition at line 181 of file sparse_str.c.

10.90.2.3 [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.

10.90.2.4 void fasp_dstr_free ([dSTRmat](#) * *A*)

Free STR sparse matrix data memeory space.

Parameters

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

Author

Shiquan Zhang, Xiaozhe Hu

Date

05/17/2010

Definition at line 152 of file sparse_str.c.

10.90.2.5 void fasp_dstr_null (dSTRmat * *A*)

Initialize sparse matrix on structured grid.

Parameters

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

Author

Shiquan Zhang, Xiaozhe Hu

Date

05/17/2010

Definition at line 25 of file sparse_str.c.

10.91 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

10.91.1 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

10.91.2 Function Documentation

10.91.2.1 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.

10.91.2.2 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.

10.91.2.3 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.

10.91.2.4 void void fasp_sparse_aplbms_ (INT * *ia*, INT * *ja*, INT * *ib*, INT * *jb*, INT * *nab*, INT * *mab*, INT * *ic*, INT * *jc*)

Addition of two sparse matrices: calculating the nonzero structure of the result if *jc* is not null. if *jc* is null only finds num of nonzeros.

Parameters

<i>ia</i>	array of row pointers 1st summand
<i>ja</i>	array of column indices 1st summand
<i>ib</i>	array of row pointers 2nd summand
<i>jb</i>	array of column indices 2nd summand
<i>nab</i>	number of rows
<i>mab</i>	number of cols
<i>ic</i>	array of row pointers in the result (this is also computed here again, so that we can have a stand alone call of this routine, if for some reason the number of nonzeros in the result is known)
<i>jc</i>	array of column indices in the result $c=a+b$

Definition at line 359 of file sparse_util.c.

10.91.2.5 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.

10.91.2.6 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.

10.91.2.7 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.

10.91.2.8 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 $R*A*P$ for known nonzero structure of the result the result is stored in iac,jac,ac!

Definition at line 788 of file sparse_util.c.

10.91.2.9 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 $R*A*P$, 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 $R*A*P$. 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.

10.91.2.10 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.

10.91.2.11 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.

10.91.2.12 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.

10.91.2.13 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>v</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.

10.92 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_spgs (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.

10.92.1 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_{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

10.92.2 Function Documentation

10.92.2.1 `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 block_dCSRmat : 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.

10.92.2.2 `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 dBSRmat : 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.

10.92.2.3 **INT** fasp_solver_dcsr_spbcgs (**dCSRmat** * *A*, **dvector** * *b*, **dvector** * *u*, **precond** * *pc*, **const REAL** *tol*, **const INT** *MaxIt*, **const SHORT** *stop_type*, **const SHORT** *prtlvl*)

Preconditioned BiCGstab method for solving $Au=b$ with safety net.

Parameters

<i>A</i>	Pointer to dCSRmat : 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.

10.92.2.4 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 dSTRmat : 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.

10.93 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.

10.93.1 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

10.93.2 Function Documentation

10.93.2.1 `INT fasp_solver_bdcsr_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 block_dCSRmat : 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.

10.93.2.2 `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 dCSRmat : 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.

10.93.2.3 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 dSTRmat : 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.

10.94 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.

10.94.1 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

10.94.2 Function Documentation

10.94.2.1 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 block_dCSRmat : 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.

10.94.2.2 `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 dBSRmat : 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.

10.94.2.3 `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 dCSRmat : 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.

10.94.2.4 `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 dSTRmat : 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.

10.95 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.

10.95.1 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_{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

10.95.2 Function Documentation

10.95.2.1 **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 block_dCSRmat : 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.

10.95.2.2 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 dCSRmat : 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.

10.95.2.3 `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 dSTRmat : 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.

10.96 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.

10.96.1 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

10.96.2 Function Documentation

10.96.2.1 `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 block_dCSRmat : 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.

10.96.2.2 `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 dBSRmat : 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.

10.96.2.3 `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 dCSRmat : 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.

10.96.2.4 **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 dSTRmat : 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.

10.97 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

10.97.1 Detailed Description

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

10.97.2 Function Documentation

10.97.2.1 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.

10.97.2.2 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.

10.97.3 Variable Documentation

10.97.3.1 INT THDs_AMG_GS =0

AMG GS smoothing threads

Definition at line 107 of file threads.c.

10.97.3.2 INT THDs_CPR_gGS =0

global matrix GS smoothing threads

Definition at line 109 of file threads.c.

10.97.3.3 INT THDs_CPR_IGS =0

reservoir GS smoothing threads

Definition at line 108 of file threads.c.

10.98 timing.c File Reference

Timing subroutines.

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

Functions

- void [fasp_gettime](#) (REAL *time)
Get system time.

10.98.1 Detailed Description

Timing subroutines.

10.98.2 Function Documentation

10.98.2.1 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.

10.99 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 $D^{-1/2}b$.

10.99.1 Detailed Description

Simple operations for vectors.

Note

All structures should be initialized before usage.

10.99.2 Function Documentation

10.99.2.1 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.

10.99.2.2 `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.

10.99.2.3 `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.

10.99.2.4 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.

10.99.2.5 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.

10.99.2.6 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.

10.99.2.7 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.

10.99.2.8 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.

10.99.2.9 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.

10.99.2.10 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.

10.99.2.11 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.

10.99.2.12 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.

10.99.2.13 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.

10.99.2.14 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.

10.100 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)

10.100.1 Detailed Description

Wrappers for accessing functions by advanced users.

10.100.2 Function Documentation

10.100.2.1 `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.

10.100.2.2 `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.

10.100.2.3 **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.

10.100.2.4 **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, [70](#)
 precond_sweeping_data, [73](#)

A_diag

 precond_block_data, [58](#)

ABS

 fasp.h, [161](#)

AMG_ILU_levels

 input_param, [44](#)

AMG_Schwarz_levels

 input_param, [46](#)

AMG_aggregation_type

 input_param, [43](#)

AMG_aggressive_level

 input_param, [43](#)

AMG_aggressive_path

 input_param, [44](#)

AMG_amli_degree

 input_param, [44](#)

AMG_coarse_dof

 input_param, [44](#)

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, [45](#)

AMG_max_aggregation

 input_param, [45](#)

AMG_max_row_sum

 input_param, [45](#)

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, [46](#)

AMG_relaxation

 input_param, [46](#)

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, [47](#)

AMG_truncation_threshold

 input_param, [47](#)

AMG_type

 input_param, [47](#)

AMLI_CYCLE

 fasp_const.h, [172](#)

ASCEND

 fasp_const.h, [172](#)

Abcsr

 precond_block_data, [58](#)

Ai

 precond_sweeping_data, [73](#)

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, 58
- 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_aAxpby, 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_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_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_aAxpby, 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, [29](#)
 - fasp_block.h, [168](#)
- block_dvector, [29](#)
 - fasp_block.h, [168](#)
- block_iCSRmat, [30](#)
 - 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, [33](#)
 - fasp.h, [164](#)
- dCSRmat, [33](#)
 - fasp.h, [164](#)
- dCSRmat, [34](#)
 - fasp.h, [164](#)
- dCSRmat2SAMGInput
 - interface_samg.c, [218](#)
- DESCEND
 - fasp_const.h, [174](#)
- DIAGONAL_PREF
 - fasp.h, [161](#)
- DLMALLOC
 - fasp.h, [161](#)
- dSTRmat, [35](#)
 - fasp.h, [164](#)
- ddenmat, [35](#)
 - fasp.h, [164](#)
- diag
 - precond_block_reservoir_data, [60](#)
- diaginv
 - precond_FASP_blkoi_data, [70](#)
 - precond_block_reservoir_data, [60](#)
- diaginv_S
 - precond_FASP_blkoi_data, [70](#)
- diaginv_noscale
 - precond_FASP_blkoi_data, [70](#)
- diaginvS
 - precond_block_reservoir_data, [61](#)
- dlength
 - io.c, [245](#)
- doxygen.h, [156](#)
- dvector, [36](#)
 - fasp.h, [164](#)
- dvector2SAMGInput
 - interface_samg.c, [218](#)
- 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, 38
- efather
 - grid2d, 38
- 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, 456
- 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
- FPPFIRST
 - 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, [282](#)
- fasp_Schwarz_data_free
 - init.c, [213](#)
- fasp_Schwarz_get_block_matrix
 - schwarz_setup.c, [360](#)
- fasp_Schwarz_setup
 - schwarz_setup.c, [360](#)
- 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, [210](#)
- fasp_amg_data_bsr_free
 - init.c, [210](#)
- fasp_amg_data_create
 - init.c, [211](#)
- fasp_amg_data_free
 - init.c, [211](#)
- fasp_amg_interp
 - interpolation.c, [221](#)
- fasp_amg_interp1
 - interpolation.c, [222](#)
- fasp_amg_interp_em
 - interpolation_em.c, [223](#)
- fasp_amg_interp_trunc
 - interpolation.c, [222](#)
- 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, [278](#)
- fasp_aux_dQuickSortIndex
 - ordering.c, [278](#)
- fasp_aux_givens
 - givens.c, [196](#)
- fasp_aux_iQuickSort
 - ordering.c, [278](#)
- fasp_aux_iQuickSortIndex
 - ordering.c, [280](#)
- fasp_aux_merge
 - ordering.c, [280](#)
- fasp_aux_msor
 - ordering.c, [281](#)
- fasp_aux_unique
 - ordering.c, [281](#)

fasp_bdcsr_free
 sparse_block.c, [396](#)
 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, [358](#)
 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, [364](#)
- 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, [362](#)
- fasp_blas_smat_inv_nc
smat.c, [362](#)
- fasp_blas_smat_inv_nc2
smat.c, [362](#)
- fasp_blas_smat_inv_nc3
smat.c, [363](#)
- fasp_blas_smat_inv_nc4
smat.c, [363](#)
- fasp_blas_smat_inv_nc5
smat.c, [363](#)
- fasp_blas_smat_inv_nc7
smat.c, [364](#)
- fasp_blas_smat_invp_nc
smat.c, [364](#)
- 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_m xv
blas_smat.c, [127](#)
- fasp_blas_smat_m xv_nc2
blas_smat.c, [127](#)
- fasp_blas_smat_m xv_nc3
blas_smat.c, [129](#)
- fasp_blas_smat_m xv_nc5
blas_smat.c, [129](#)
- fasp_blas_smat_m xv_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, [272](#)
- 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_PARDISO, [186](#)
- SOLVER_SBiCGstab, [186](#)
- SOLVER_SCG, [186](#)
- SOLVER_SGCG, [186](#)
- SOLVER_SGMRES, [186](#)
- SOLVER_SMinRes, [186](#)
- SOLVER_SUPERLU, [186](#)
- SOLVER_SVFGMRES, [186](#)
- SOLVER_SVGMRES, [187](#)
- 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, [188](#)
- UA_AMG, [188](#)
- UNPT, [188](#)
- USERDEFINED, [188](#)
- V_CYCLE, [188](#)
- VMB, [188](#)
- W_CYCLE, [188](#)
- fasp_dbsr_Linfinity_dcsr
 - sparse_block.c, [398](#)
- fasp_dbsr_alloc
 - sparse_bsr.c, [400](#)
- fasp_dbsr_cp
 - sparse_bsr.c, [400](#)
- fasp_dbsr_create
 - sparse_bsr.c, [401](#)
- fasp_dbsr_diagLU
 - sparse_bsr.c, [403](#)
- fasp_dbsr_diagLU2
 - sparse_bsr.c, [404](#)
- fasp_dbsr_diaginv
 - sparse_bsr.c, [401](#)
- fasp_dbsr_diaginv2
 - sparse_bsr.c, [402](#)
- fasp_dbsr_diaginv3
 - sparse_bsr.c, [402](#)
- fasp_dbsr_diaginv4
 - sparse_bsr.c, [403](#)
- fasp_dbsr_diagpref
 - sparse_bsr.c, [404](#)
- fasp_dbsr_free
 - sparse_bsr.c, [405](#)
- fasp_dbsr_getblk
 - sparse_block.c, [397](#)
- fasp_dbsr_getblk_dcsr
 - sparse_block.c, [397](#)
- fasp_dbsr_getdiag
 - sparse_bsr.c, [405](#)
- fasp_dbsr_getdiaginv
 - sparse_bsr.c, [406](#)
- fasp_dbsr_null
 - sparse_bsr.c, [406](#)
- fasp_dbsr_plot
 - graphics.c, [203](#)
- fasp_dbsr_print
 - io.c, [226](#)
- fasp_dbsr_read
 - io.c, [226](#)
- fasp_dbsr_subplot
 - graphics.c, [204](#)
- fasp_dbsr_trans
 - sparse_bsr.c, [406](#)
- fasp_dbsr_write
 - io.c, [227](#)
- fasp_dbsr_write_coo
 - io.c, [227](#)
- fasp_dcoo1_read
 - io.c, [228](#)
- fasp_dcoo_alloc
 - sparse_coo.c, [407](#)
- fasp_dcoo_create
 - sparse_coo.c, [408](#)
- fasp_dcoo_free
 - sparse_coo.c, [408](#)
- fasp_dcoo_print
 - io.c, [228](#)
- fasp_dcoo_read
 - io.c, [229](#)
- fasp_dcoo_shift
 - sparse_coo.c, [408](#)
- fasp_dcoo_shift_read
 - io.c, [229](#)
- fasp_dcoo_write
 - io.c, [230](#)
- fasp_dcsr_CMK_order
 - ordering.c, [282](#)
- fasp_dcsr_RCMK_order
 - ordering.c, [283](#)
- fasp_dcsr_Schwarz_backward_smoother
 - schwarz_setup.c, [359](#)
- fasp_dcsr_Schwarz_forward_smoother
 - schwarz_setup.c, [359](#)
- fasp_dcsr_alloc
 - sparse_csr.c, [410](#)
- fasp_dcsr_compress
 - sparse_csr.c, [411](#)
- fasp_dcsr_compress_inplace
 - sparse_csr.c, [411](#)
- fasp_dcsr_cp
 - sparse_csr.c, [412](#)
- fasp_dcsr_create
 - sparse_csr.c, [412](#)

`fasp_dcsr_diagpref`
 `sparse_csr.c`, 412

`fasp_dcsr_eig`
 `eigen.c`, 156

`fasp_dcsr_free`
 `sparse_csr.c`, 414

`fasp_dcsr_getblk`
 `sparse_block.c`, 398

`fasp_dcsr_getcol`
 `sparse_csr.c`, 414

`fasp_dcsr_getdiag`
 `sparse_csr.c`, 415

`fasp_dcsr_multicoloring`
 `sparse_csr.c`, 415

`fasp_dcsr_null`
 `sparse_csr.c`, 416

`fasp_dcsr_perm`
 `sparse_csr.c`, 416

`fasp_dcsr_permz`
 `sparse_csr.c`, 416

`fasp_dcsr_plot`
 `graphics.c`, 204

`fasp_dcsr_print`
 `io.c`, 230

`fasp_dcsr_read`
 `io.c`, 231

`fasp_dcsr_regdiag`
 `sparse_csr.c`, 417

`fasp_dcsr_shift`
 `sparse_csr.c`, 417

`fasp_dcsr_sort`
 `sparse_csr.c`, 418

`fasp_dcsr_sortz`
 `sparse_csr.c`, 418

`fasp_dcsr_subplot`
 `graphics.c`, 205

`fasp_dcsr_symdiagscale`
 `sparse_csr.c`, 418

`fasp_dcsr_sympat`
 `sparse_csr.c`, 419

`fasp_dcsr_trans`
 `sparse_csr.c`, 419

`fasp_dcsr_transz`
 `sparse_csr.c`, 420

`fasp_dcsr_write_coo`
 `io.c`, 231

`fasp_dcsl_create`
 `sparse_csr.c`, 424

`fasp_dcsl_free`
 `sparse_csr.c`, 424

`fasp_dcsrvec1_read`
 `io.c`, 231

`fasp_dcsrvec1_write`
 `io.c`, 232

`fasp_dcsrvec2_read`
 `io.c`, 233

`fasp_dcsrvec2_write`
 `io.c`, 233

`fasp_dmtx_read`
 `io.c`, 234

`fasp_dmtxsym_read`
 `io.c`, 234

`fasp_dstr_alloc`
 `sparse_str.c`, 425

`fasp_dstr_cp`
 `sparse_str.c`, 425

`fasp_dstr_create`
 `sparse_str.c`, 427

`fasp_dstr_diagscale`
 `blas_str.c`, 139

`fasp_dstr_free`
 `sparse_str.c`, 427

`fasp_dstr_null`
 `sparse_str.c`, 428

`fasp_dstr_print`
 `io.c`, 236

`fasp_dstr_read`
 `io.c`, 236

`fasp_dstr_write`
 `io.c`, 237

`fasp_dvec_alloc`
 `vec.c`, 459

`fasp_dvec_cp`
 `vec.c`, 460

`fasp_dvec_create`
 `vec.c`, 460

`fasp_dvec_free`
 `vec.c`, 460

`fasp_dvec_isnan`
 `vec.c`, 462

`fasp_dvec_maxdiff`
 `vec.c`, 462

`fasp_dvec_null`
 `vec.c`, 463

`fasp_dvec_print`
 `io.c`, 237

`fasp_dvec_rand`
 `vec.c`, 463

`fasp_dvec_read`
 `io.c`, 238

`fasp_dvec_set`
 `vec.c`, 464

`fasp_dvec_symdiagscale`
 `vec.c`, 464

`fasp_dvec_write`
 `io.c`, 238

`fasp_dvecind_read`
 `io.c`, 238

fasp_dvecind_write
io.c, [239](#)

fasp_famg_solve
amg_solve.c, [87](#)

fasp_format_bdcscr_dcsr
formats.c, [190](#)

fasp_format_dbsr_dcoo
formats.c, [191](#)

fasp_format_dbsr_dcsr
formats.c, [191](#)

fasp_format_dcoo_dcsr
formats.c, [191](#)

fasp_format_dcsr_dbsr
formats.c, [193](#)

fasp_format_dcsr_dcoo
formats.c, [193](#)

fasp_format_dcsrl_dcsr
formats.c, [194](#)

fasp_format_dstr_dbsr
formats.c, [194](#)

fasp_format_dstr_dcsr
formats.c, [195](#)

fasp_fwrapper_amg_
wrapper.c, [467](#)

fasp_fwrapper_krylov_amg_
wrapper.c, [467](#)

fasp_gauss2d
quadrature.c, [356](#)

fasp_generate_diaginv_block
smoother_str.c, [388](#)

fasp_gettime
timing.c, [457](#)

fasp_grid2d_plot
graphics.c, [205](#)

fasp_hb_read
io.c, [239](#)

fasp_iarray_cp
array.c, [93](#)

fasp_iarray_set
array.c, [93](#)

fasp_icsr_cp
sparse_csr.c, [420](#)

fasp_icsr_create
sparse_csr.c, [421](#)

fasp_icsr_free
sparse_csr.c, [421](#)

fasp_icsr_null
sparse_csr.c, [421](#)

fasp_icsr_trans
sparse_csr.c, [423](#)

fasp_iden_free
smat.c, [365](#)

fasp_ilu_data_alloc
init.c, [212](#)

fasp_ilu_data_free
init.c, [212](#)

fasp_ilu_data_null
init.c, [212](#)

fasp_ilu_dbsr_setup
ilu_setup_bsr.c, [206](#)

fasp_ilu_dcsr_setup
ilu_setup_csr.c, [207](#)

fasp_ilu_dstr_setup0
ilu_setup_str.c, [208](#)

fasp_ilu_dstr_setup1
ilu_setup_str.c, [209](#)

fasp_ivec_alloc
vec.c, [464](#)

fasp_ivec_create
vec.c, [465](#)

fasp_ivec_free
vec.c, [465](#)

fasp_ivec_print
io.c, [240](#)

fasp_ivec_read
io.c, [240](#)

fasp_ivec_set
vec.c, [465](#)

fasp_ivec_write
io.c, [241](#)

fasp_ivecind_read
io.c, [241](#)

fasp_matrix_read
io.c, [242](#)

fasp_matrix_read_bin
io.c, [242](#)

fasp_matrix_write
io.c, [243](#)

fasp_mem_calloc
memory.c, [269](#)

fasp_mem_check
memory.c, [269](#)

fasp_mem_dcsr_check
memory.c, [269](#)

fasp_mem_free
memory.c, [270](#)

fasp_mem_iludata_check
memory.c, [270](#)

fasp_mem_realloc
memory.c, [271](#)

fasp_mem_usage
memory.c, [271](#)

fasp_param_Schwarz_init
parameters.c, [288](#)

fasp_param_Schwarz_print
parameters.c, [290](#)

fasp_param_Schwarz_set
parameters.c, [290](#)

fasp_param_amg_init
 parameters.c, [284](#)
 fasp_param_amg_print
 parameters.c, [285](#)
 fasp_param_amg_set
 parameters.c, [285](#)
 fasp_param_amg_to_prec
 parameters.c, [285](#)
 fasp_param_amg_to_prec_bsr
 parameters.c, [286](#)
 fasp_param_check
 input.c, [214](#)
 fasp_param_ilu_init
 parameters.c, [286](#)
 fasp_param_ilu_print
 parameters.c, [286](#)
 fasp_param_ilu_set
 parameters.c, [287](#)
 fasp_param_init
 parameters.c, [287](#)
 fasp_param_input
 input.c, [215](#)
 fasp_param_input_init
 parameters.c, [287](#)
 fasp_param_prec_to_amg
 parameters.c, [288](#)
 fasp_param_prec_to_amg_bsr
 parameters.c, [288](#)
 fasp_param_set
 parameters.c, [290](#)
 fasp_param_solver_init
 parameters.c, [291](#)
 fasp_param_solver_print
 parameters.c, [291](#)
 fasp_param_solver_set
 parameters.c, [291](#)
 fasp_poisson_fgmg_1D
 gmg_poisson.c, [197](#)
 fasp_poisson_fgmg_2D
 gmg_poisson.c, [197](#)
 fasp_poisson_fgmg_3D
 gmg_poisson.c, [198](#)
 fasp_poisson_gmg_1D
 gmg_poisson.c, [198](#)
 fasp_poisson_gmg_2D
 gmg_poisson.c, [199](#)
 fasp_poisson_gmg_3D
 gmg_poisson.c, [199](#)
 fasp_poisson_pcg_gmg_1D
 gmg_poisson.c, [200](#)
 fasp_poisson_pcg_gmg_2D
 gmg_poisson.c, [200](#)
 fasp_poisson_pcg_gmg_3D
 gmg_poisson.c, [202](#)
 fasp_precond_Schwarz
 precond_csr.c, [341](#)
 fasp_precond_amg
 precond_csr.c, [337](#)
 fasp_precond_amg_nk
 precond_csr.c, [338](#)
 fasp_precond_amli
 precond_csr.c, [338](#)
 fasp_precond_block_SGS_3
 precond_bcsr.c, [325](#)
 fasp_precond_block_SGS_3_amg
 precond_bcsr.c, [327](#)
 fasp_precond_block_diag_3
 precond_bcsr.c, [322](#)
 fasp_precond_block_diag_3_amg
 precond_bcsr.c, [323](#)
 fasp_precond_block_diag_4
 precond_bcsr.c, [323](#)
 fasp_precond_block_lower_3
 precond_bcsr.c, [323](#)
 fasp_precond_block_lower_3_amg
 precond_bcsr.c, [325](#)
 fasp_precond_block_lower_4
 precond_bcsr.c, [325](#)
 fasp_precond_block_upper_3
 precond_bcsr.c, [327](#)
 fasp_precond_block_upper_3_amg
 precond_bcsr.c, [327](#)
 fasp_precond_data_null
 init.c, [213](#)
 fasp_precond_dbsr_amg
 precond_bsr.c, [330](#)
 fasp_precond_dbsr_amg_nk
 precond_bsr.c, [330](#)
 fasp_precond_dbsr_diag
 precond_bsr.c, [331](#)
 fasp_precond_dbsr_diag_nc2
 precond_bsr.c, [331](#)
 fasp_precond_dbsr_diag_nc3
 precond_bsr.c, [332](#)
 fasp_precond_dbsr_diag_nc5
 precond_bsr.c, [332](#)
 fasp_precond_dbsr_diag_nc7
 precond_bsr.c, [334](#)
 fasp_precond_dbsr_ilu
 precond_bsr.c, [334](#)
 fasp_precond_dbsr_nl_amli
 precond_bsr.c, [336](#)
 fasp_precond_diag
 precond_csr.c, [338](#)
 fasp_precond_dstr_blockgs
 precond_str.c, [343](#)
 fasp_precond_dstr_diag
 precond_str.c, [343](#)

fasp_precond_dstr_ilu0
 precond_str.c, [343](#)

fasp_precond_dstr_ilu0_backward
 precond_str.c, [344](#)

fasp_precond_dstr_ilu0_forward
 precond_str.c, [344](#)

fasp_precond_dstr_ilu1
 precond_str.c, [344](#)

fasp_precond_dstr_ilu1_backward
 precond_str.c, [346](#)

fasp_precond_dstr_ilu1_forward
 precond_str.c, [346](#)

fasp_precond_famg
 precond_csr.c, [339](#)

fasp_precond_free
 precond_csr.c, [339](#)

fasp_precond_ilu
 precond_csr.c, [339](#)

fasp_precond_ilu_backward
 precond_csr.c, [340](#)

fasp_precond_ilu_forward
 precond_csr.c, [340](#)

fasp_precond_nl_amli
 precond_csr.c, [341](#)

fasp_precond_null
 init.c, [213](#)

fasp_precond_setup
 precond_csr.c, [341](#)

fasp_precond_sweeping
 precond_bcsr.c, [329](#)

fasp_quad2d
 quadrature.c, [357](#)

fasp_set_GS_threads
 threads.c, [456](#)

fasp_smat_identity
 smat.c, [365](#)

fasp_smat_identity_nc2
 smat.c, [365](#)

fasp_smat_identity_nc3
 smat.c, [367](#)

fasp_smat_identity_nc5
 smat.c, [367](#)

fasp_smat_identity_nc7
 smat.c, [367](#)

fasp_smat_lu_decomp
 lu.c, [266](#)

fasp_smat_lu_solve
 lu.c, [267](#)

fasp_smoother_dbsr_gs
 smoother_bsr.c, [369](#)

fasp_smoother_dbsr_gs1
 smoother_bsr.c, [369](#)

fasp_smoother_dbsr_gs_ascend
 smoother_bsr.c, [370](#)

fasp_smoother_dbsr_gs_ascend1
 smoother_bsr.c, [370](#)

fasp_smoother_dbsr_gs_descend
 smoother_bsr.c, [371](#)

fasp_smoother_dbsr_gs_descend1
 smoother_bsr.c, [371](#)

fasp_smoother_dbsr_gs_order1
 smoother_bsr.c, [372](#)

fasp_smoother_dbsr_gs_order2
 smoother_bsr.c, [372](#)

fasp_smoother_dbsr_ilu
 smoother_bsr.c, [373](#)

fasp_smoother_dbsr_jacobi
 smoother_bsr.c, [373](#)

fasp_smoother_dbsr_jacobi1
 smoother_bsr.c, [373](#)

fasp_smoother_dbsr_jacobi_setup
 smoother_bsr.c, [375](#)

fasp_smoother_dbsr_sor
 smoother_bsr.c, [375](#)

fasp_smoother_dbsr_sor1
 smoother_bsr.c, [376](#)

fasp_smoother_dbsr_sor_ascend
 smoother_bsr.c, [376](#)

fasp_smoother_dbsr_sor_descend
 smoother_bsr.c, [377](#)

fasp_smoother_dbsr_sor_order
 smoother_bsr.c, [377](#)

fasp_smoother_dcsr_L1diag
 smoother_csr.c, [382](#)

fasp_smoother_dcsr_gs
 smoother_csr.c, [379](#)

fasp_smoother_dcsr_gs_cf
 smoother_csr.c, [379](#)

fasp_smoother_dcsr_gs_rb3d
 smoother_csr.c, [380](#)

fasp_smoother_dcsr_gscr
 smoother_csr_cr.c, [384](#)

fasp_smoother_dcsr_ilu
 smoother_csr.c, [380](#)

fasp_smoother_dcsr_jacobi
 smoother_csr.c, [381](#)

fasp_smoother_dcsr_kaczmarz
 smoother_csr.c, [381](#)

fasp_smoother_dcsr_poly
 smoother_csr_poly.c, [386](#)

fasp_smoother_dcsr_poly_old
 smoother_csr_poly.c, [386](#)

fasp_smoother_dcsr_sgs
 smoother_csr.c, [382](#)

fasp_smoother_dcsr_sor
 smoother_csr.c, [383](#)

fasp_smoother_dcsr_sor_cf
 smoother_csr.c, [383](#)

fasp_smoother_dstr_gs
 smoother_str.c, [388](#)
 fasp_smoother_dstr_gs1
 smoother_str.c, [388](#)
 fasp_smoother_dstr_gs_ascend
 smoother_str.c, [389](#)
 fasp_smoother_dstr_gs_cf
 smoother_str.c, [389](#)
 fasp_smoother_dstr_gs_descend
 smoother_str.c, [390](#)
 fasp_smoother_dstr_gs_order
 smoother_str.c, [390](#)
 fasp_smoother_dstr_jacobi
 smoother_str.c, [391](#)
 fasp_smoother_dstr_jacobi1
 smoother_str.c, [391](#)
 fasp_smoother_dstr_schwarz
 smoother_str.c, [392](#)
 fasp_smoother_dstr_sor
 smoother_str.c, [392](#)
 fasp_smoother_dstr_sor1
 smoother_str.c, [392](#)
 fasp_smoother_dstr_sor_ascend
 smoother_str.c, [393](#)
 fasp_smoother_dstr_sor_cf
 smoother_str.c, [393](#)
 fasp_smoother_dstr_sor_descend
 smoother_str.c, [394](#)
 fasp_smoother_dstr_sor_order
 smoother_str.c, [394](#)
 fasp_solver_amg
 amg.c, [77](#)
 fasp_solver_amli
 amlirecur.c, [88](#)
 fasp_solver_bdcscr_itsolver
 itsolver_bcsr.c, [246](#)
 fasp_solver_bdcscr_krylov
 itsolver_bcsr.c, [246](#)
 fasp_solver_bdcscr_krylov_block_3
 itsolver_bcsr.c, [247](#)
 fasp_solver_bdcscr_krylov_block_4
 itsolver_bcsr.c, [247](#)
 fasp_solver_bdcscr_krylov_sweeping
 itsolver_bcsr.c, [248](#)
 fasp_solver_bdcscr_pbcgs
 pbcgs.c, [293](#)
 fasp_solver_bdcscr_pcg
 pcg.c, [300](#)
 fasp_solver_bdcscr_pgmres
 pgmres.c, [311](#)
 fasp_solver_bdcscr_pminres
 pminres.c, [317](#)
 fasp_solver_bdcscr_pvfgmres
 pvfgmres.c, [347](#)
 fasp_solver_bdcscr_pvgmres
 pvgmres.c, [351](#)
 fasp_solver_bdcscr_spbcgs
 spbcgs.c, [437](#)
 fasp_solver_bdcscr_spcg
 spcg.c, [442](#)
 fasp_solver_bdcscr_spgmres
 spgmres.c, [444](#)
 fasp_solver_bdcscr_spmminres
 spmminres.c, [449](#)
 fasp_solver_bdcscr_spvgmres
 spvgmres.c, [453](#)
 fasp_solver_dbsr_itsolver
 itsolver_bsr.c, [249](#)
 fasp_solver_dbsr_krylov
 itsolver_bsr.c, [250](#)
 fasp_solver_dbsr_krylov_amg
 itsolver_bsr.c, [250](#)
 fasp_solver_dbsr_krylov_amg_nk
 itsolver_bsr.c, [251](#)
 fasp_solver_dbsr_krylov_diag
 itsolver_bsr.c, [251](#)
 fasp_solver_dbsr_krylov_ilu
 itsolver_bsr.c, [252](#)
 fasp_solver_dbsr_krylov_nk_amg
 itsolver_bsr.c, [252](#)
 fasp_solver_dbsr_pbcgs
 pbcgs.c, [294](#)
 fasp_solver_dbsr_pcg
 pcg.c, [301](#)
 fasp_solver_dbsr_pgmres
 pgmres.c, [312](#)
 fasp_solver_dbsr_pvfgmres
 pvfgmres.c, [348](#)
 fasp_solver_dbsr_pvgmres
 pvgmres.c, [352](#)
 fasp_solver_dbsr_spbcgs
 spbcgs.c, [437](#)
 fasp_solver_dbsr_spgmres
 spgmres.c, [445](#)
 fasp_solver_dbsr_spvgmres
 spvgmres.c, [453](#)
 fasp_solver_dcsr_itsolver
 itsolver_csr.c, [254](#)
 fasp_solver_dcsr_krylov
 itsolver_csr.c, [254](#)
 fasp_solver_dcsr_krylov_Schwarz
 itsolver_csr.c, [257](#)
 fasp_solver_dcsr_krylov_amg
 itsolver_csr.c, [255](#)
 fasp_solver_dcsr_krylov_amg_nk
 itsolver_csr.c, [255](#)
 fasp_solver_dcsr_krylov_diag
 itsolver_csr.c, [256](#)

fasp_solver_dcsr_krylov_ilu
 itsolver_csr.c, [256](#)

fasp_solver_dcsr_krylov_ilu_M
 itsolver_csr.c, [257](#)

fasp_solver_dcsr_pbcgs
 pbcgs.c, [295](#)

fasp_solver_dcsr_pcg
 pcg.c, [301](#)

fasp_solver_dcsr_pgcg
 pgcg.c, [306](#)

fasp_solver_dcsr_pgcr
 pgcr.c, [309](#)

fasp_solver_dcsr_pgcr1
 pgcr.c, [310](#)

fasp_solver_dcsr_pgmres
 pgmres.c, [313](#)

fasp_solver_dcsr_pminres
 pminres.c, [317](#)

fasp_solver_dcsr_pvfgmres
 pvfgmres.c, [349](#)

fasp_solver_dcsr_pvgmres
 pvgmres.c, [352](#)

fasp_solver_dcsr_spbcs
 spbcgs.c, [438](#)

fasp_solver_dcsr_spcg
 spcg.c, [442](#)

fasp_solver_dcsr_spgmres
 spgmres.c, [445](#)

fasp_solver_dcsr_spmminres
 spmminres.c, [450](#)

fasp_solver_dcsr_spvgmres
 spvgmres.c, [454](#)

fasp_solver_dstr_itsolver
 itsolver_str.c, [263](#)

fasp_solver_dstr_krylov
 itsolver_str.c, [263](#)

fasp_solver_dstr_krylov_blockgs
 itsolver_str.c, [263](#)

fasp_solver_dstr_krylov_diag
 itsolver_str.c, [265](#)

fasp_solver_dstr_krylov_ilu
 itsolver_str.c, [265](#)

fasp_solver_dstr_pbcgs
 pbcgs.c, [295](#)

fasp_solver_dstr_pcg
 pcg.c, [303](#)

fasp_solver_dstr_pgmres
 pgmres.c, [313](#)

fasp_solver_dstr_pminres
 pminres.c, [318](#)

fasp_solver_dstr_pvgmres
 pvgmres.c, [354](#)

fasp_solver_dstr_spbcs
 spbcgs.c, [438](#)

fasp_solver_dstr_spcg
 spcg.c, [443](#)

fasp_solver_dstr_spgmres
 spgmres.c, [447](#)

fasp_solver_dstr_spmminres
 spmminres.c, [450](#)

fasp_solver_dstr_spvgmres
 spvgmres.c, [454](#)

fasp_solver_famg
 famg.c, [157](#)

fasp_solver_fmecycle
 fmecycle.c, [189](#)

fasp_solver_itsolver
 itsolver_mf.c, [260](#)

fasp_solver_itsolver_init
 itsolver_mf.c, [261](#)

fasp_solver_krylov
 itsolver_mf.c, [261](#)

fasp_solver_mgcycle
 mgcycle.c, [275](#)

fasp_solver_mgcycle_bsr
 mgcycle.c, [275](#)

fasp_solver_mgrecur
 mgrecur.c, [276](#)

fasp_solver_mumps
 interface_mumps.c, [216](#)

fasp_solver_mumps_steps
 interface_mumps.c, [216](#)

fasp_solver_nl_amli
 amlirecur.c, [89](#)

fasp_solver_nl_amli_bsr
 amlirecur.c, [89](#)

fasp_solver_pardiso
 interface_pardiso.c, [217](#)

fasp_solver_pbcgs
 pbcgs_mf.c, [297](#)

fasp_solver_pcg
 pcg_mf.c, [305](#)

fasp_solver_pgcg
 pgcg_mf.c, [307](#)

fasp_solver_pgmres
 pgmres_mf.c, [315](#)

fasp_solver_pminres
 pminres_mf.c, [320](#)

fasp_solver_pvfgmres
 pvfgmres_mf.c, [350](#)

fasp_solver_pvgmres
 pvgmres_mf.c, [355](#)

fasp_solver_superlu
 interface_superlu.c, [219](#)

fasp_solver_umfpack
 interface_umfpack.c, [220](#)

fasp_sparse_MIS
 sparse_util.c, [432](#)

- fasp_sparse_aat_
 - sparse_util.c, [429](#)
 - fasp_sparse_abyb_
 - sparse_util.c, [430](#)
 - fasp_sparse_abybms_
 - sparse_util.c, [430](#)
 - fasp_sparse_aplbms_
 - sparse_util.c, [431](#)
 - fasp_sparse_aplusb_
 - sparse_util.c, [431](#)
 - fasp_sparse_iit_
 - sparse_util.c, [431](#)
 - fasp_sparse_rapcmp_
 - sparse_util.c, [432](#)
 - fasp_sparse_rapms_
 - sparse_util.c, [433](#)
 - fasp_sparse_wta_
 - sparse_util.c, [433](#)
 - fasp_sparse_wtams_
 - sparse_util.c, [434](#)
 - fasp_sparse_ytx_
 - sparse_util.c, [434](#)
 - fasp_sparse_ytxbig_
 - sparse_util.c, [435](#)
 - fasp_vector_read
 - io.c, [243](#)
 - fasp_vector_write
 - io.c, [244](#)
 - fasp_wrapper_dbsr_krylov_amg
 - wrapper.c, [468](#)
 - fasp_wrapper_dcoo_dbsr_krylov_amg
 - wrapper.c, [468](#)
 - fmgcycle.c, [189](#)
 - fasp_solver_fmgcycle, [189](#)
 - formats.c, [189](#)
 - fasp_format_bdcsr_dcsr, [190](#)
 - fasp_format_dbsr_dcoo, [191](#)
 - fasp_format_dbsr_dcsr, [191](#)
 - fasp_format_dcoo_dcsr, [191](#)
 - fasp_format_dcsr_dbsr, [193](#)
 - fasp_format_dcsr_dcoo, [193](#)
 - fasp_format_dcsr_dcsr, [194](#)
 - fasp_format_dstr_dbsr, [194](#)
 - fasp_format_dstr_dcsr, [195](#)
 - G0PT
 - fasp_const.h, [177](#)
 - GE
 - fasp.h, [162](#)
 - GT
 - fasp.h, [162](#)
 - givens.c, [195](#)
 - fasp_aux_givens, [196](#)
 - gmg_poisson.c, [196](#)
 - fasp_poisson_fgm_1D, [197](#)
 - fasp_poisson_fgm_2D, [197](#)
 - fasp_poisson_fgm_3D, [198](#)
 - fasp_poisson_gmg_1D, [198](#)
 - fasp_poisson_gmg_2D, [199](#)
 - fasp_poisson_gmg_3D, [199](#)
 - fasp_poisson_pcg_gmg_1D, [200](#)
 - fasp_poisson_pcg_gmg_2D, [200](#)
 - fasp_poisson_pcg_gmg_3D, [202](#)
 - graphics.c, [203](#)
 - fasp_dbsr_plot, [203](#)
 - fasp_dbsr_subplot, [204](#)
 - fasp_dcsr_plot, [204](#)
 - fasp_dcsr_subplot, [205](#)
 - fasp_grid2d_plot, [205](#)
 - grid2d, [37](#)
 - e, [37](#)
 - edges, [37](#)
 - ediri, [38](#)
 - efather, [38](#)
 - fasp.h, [164](#)
 - p, [38](#)
 - pdiri, [38](#)
 - pfather, [38](#)
 - s, [38](#)
 - t, [38](#)
 - tfather, [38](#)
 - triangles, [38](#)
 - vertices, [39](#)
 - ICNTL
 - interface_mumps.c, [216](#)
 - iCOOmat, [39](#)
 - fasp.h, [164](#)
 - iCSRmat, [40](#)
 - fasp.h, [164](#)
 - ILU_data, [41](#)
 - 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, [245](#)
- ilu_setup_bsr.c, [206](#)
 - fasp_ilu_dbsr_setup, [206](#)
- ilu_setup_csr.c, [207](#)
 - fasp_ilu_dcsr_setup, [207](#)
- ilu_setup_str.c, [208](#)
 - fasp_ilu_dstr_setup0, [208](#)
 - fasp_ilu_dstr_setup1, [209](#)
- inifile
 - input_param, [47](#)
- init.c, [209](#)
 - fasp_Schwarz_data_free, [213](#)
 - fasp_amg_data_bsr_create, [210](#)
 - fasp_amg_data_bsr_free, [210](#)
 - fasp_amg_data_create, [211](#)
 - fasp_amg_data_free, [211](#)
 - fasp_ilu_data_alloc, [212](#)
 - fasp_ilu_data_free, [212](#)
 - fasp_ilu_data_null, [212](#)
 - fasp_precond_data_null, [213](#)
 - fasp_precond_null, [213](#)
- input.c, [214](#)
 - fasp_param_check, [214](#)
 - fasp_param_input, [215](#)
- input_param, [42](#)
 - AMG_ILU_levels, [44](#)
 - AMG_Schwarz_levels, [46](#)
 - AMG_aggregation_type, [43](#)
 - AMG_aggressive_level, [43](#)
 - AMG_aggressive_path, [44](#)
 - AMG_amli_degree, [44](#)
 - AMG_coarse_dof, [44](#)
 - AMG_coarse_scaling, [44](#)
 - AMG_coarse_solver, [44](#)
 - AMG_coarsening_type, [44](#)
 - AMG_cycle_type, [44](#)
 - AMG_interpolation_type, [44](#)
 - AMG_levels, [45](#)
 - AMG_max_aggregation, [45](#)
 - AMG_max_row_sum, [45](#)
 - 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, [46](#)
 - AMG_relaxation, [46](#)
 - 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, [47](#)
 - AMG_truncation_threshold, [47](#)
 - AMG_type, [47](#)
 - ILU_droptol, [47](#)
 - ILU_lfil, [47](#)
 - ILU_permtol, [47](#)
 - ILU_relax, [47](#)
 - ILU_type, [47](#)
 - inifile, [47](#)
 - itsolver_maxit, [48](#)
 - itsolver_tol, [48](#)
 - output_type, [48](#)
 - precond_type, [48](#)
 - print_level, [48](#)
 - problem_num, [48](#)
 - restart, [48](#)
 - Schwarz_blk solver, [48](#)
 - Schwarz_maxlvl, [48](#)
 - Schwarz_mmsize, [49](#)
 - Schwarz_type, [49](#)
 - solver_type, [49](#)
 - stop_type, [49](#)
 - workdir, [49](#)
- interface_mumps.c, [215](#)
 - fasp_solver_mumps, [216](#)
 - fasp_solver_mumps_steps, [216](#)
 - ICNTL, [216](#)
- interface_pardiso.c, [217](#)
 - fasp_solver_pardiso, [217](#)
- interface_samg.c, [218](#)
 - dCSRmat2SAMGInput, [218](#)
 - dvector2SAMGInput, [218](#)
- interface_superlu.c, [219](#)
 - fasp_solver_superlu, [219](#)
- interface_umfpack.c, [220](#)
 - fasp_solver_umfpack, [220](#)
- interpolation.c, [221](#)

- fasp_amg_interp, 221
 - fasp_amg_interp1, 222
 - fasp_amg_interp_trunc, 222
- interpolation_em.c, 223
 - fasp_amg_interp_em, 223
- io.c, 224
 - dlength, 245
 - fasp_dbsr_print, 226
 - fasp_dbsr_read, 226
 - fasp_dbsr_write, 227
 - fasp_dbsr_write_coo, 227
 - fasp_dcoo1_read, 228
 - fasp_dcoo_print, 228
 - fasp_dcoo_read, 229
 - fasp_dcoo_shift_read, 229
 - fasp_dcoo_write, 230
 - fasp_dcsr_print, 230
 - fasp_dcsr_read, 231
 - fasp_dcsr_write_coo, 231
 - fasp_dcsrvec1_read, 231
 - fasp_dcsrvec1_write, 232
 - fasp_dcsrvec2_read, 233
 - fasp_dcsrvec2_write, 233
 - fasp_dmtx_read, 234
 - fasp_dmtxsym_read, 234
 - fasp_dstr_print, 236
 - fasp_dstr_read, 236
 - fasp_dstr_write, 237
 - fasp_dvec_print, 237
 - fasp_dvec_read, 238
 - fasp_dvec_write, 238
 - fasp_dvecind_read, 238
 - fasp_dvecind_write, 239
 - fasp_hb_read, 239
 - fasp_ivec_print, 240
 - fasp_ivec_read, 240
 - fasp_ivec_write, 241
 - fasp_ivecind_read, 241
 - fasp_matrix_read, 242
 - fasp_matrix_read_bin, 242
 - fasp_matrix_write, 243
 - fasp_vector_read, 243
 - fasp_vector_write, 244
 - ilength, 245
- itsolver_bcsr.c, 245
 - fasp_solver_bdcsr_itsolver, 246
 - fasp_solver_bdcsr_krylov, 246
 - fasp_solver_bdcsr_krylov_block_3, 247
 - fasp_solver_bdcsr_krylov_block_4, 247
 - fasp_solver_bdcsr_krylov_sweeping, 248
- itsolver_bsr.c, 249
 - fasp_solver_dbsr_itsolver, 249
 - fasp_solver_dbsr_krylov, 250
 - fasp_solver_dbsr_krylov_amg, 250
 - fasp_solver_dbsr_krylov_amg_nk, 251
 - fasp_solver_dbsr_krylov_diag, 251
 - fasp_solver_dbsr_krylov_ilu, 252
 - fasp_solver_dbsr_krylov_nk_amg, 252
- itsolver_csr.c, 253
 - fasp_solver_dcsr_itsolver, 254
 - fasp_solver_dcsr_krylov, 254
 - fasp_solver_dcsr_krylov_Schwarz, 257
 - fasp_solver_dcsr_krylov_amg, 255
 - fasp_solver_dcsr_krylov_amg_nk, 255
 - fasp_solver_dcsr_krylov_diag, 256
 - fasp_solver_dcsr_krylov_ilu, 256
 - fasp_solver_dcsr_krylov_ilu_M, 257
- itsolver_maxit
 - input_param, 48
- itsolver_mf.c, 259
 - fasp_solver_itsolver, 260
 - fasp_solver_itsolver_init, 261
 - fasp_solver_krylov, 261
- itsolver_param, 49
 - itsolver_type, 50
 - maxit, 50
 - precond_type, 50
 - print_level, 50
 - restart, 50
 - stop_type, 50
 - tol, 50
- itsolver_str.c, 262
 - fasp_solver_dstr_itsolver, 263
 - fasp_solver_dstr_krylov, 263
 - fasp_solver_dstr_krylov_blockgs, 263
 - fasp_solver_dstr_krylov_diag, 265
 - fasp_solver_dstr_krylov_ilu, 265
- itsolver_tol
 - input_param, 48
- itsolver_type
 - itsolver_param, 50
- ivector, 51
 - 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, 58
- Link, 51
- LinkedList

- fasp.h, 165
- linked_list, 52
- ListElement
 - fasp.h, 165
- local_A
 - precond_sweeping_data, 73
- local_LU
 - precond_sweeping_data, 73
- local_index
 - precond_sweeping_data, 73
- lu.c, 266
 - fasp_smat_lu_decomp, 266
 - fasp_smat_lu_solve, 267
- 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, 53
- malloc_params, 53
- malloc_segment, 54
- malloc_state, 54
- malloc_tree_chunk, 55
- maxit
 - itsolver_param, 50
 - precond_FASP_blkoi_data, 70
- memory.c, 268
 - fasp_mem_calloc, 269
 - fasp_mem_check, 269
 - fasp_mem_dcsr_check, 269
 - fasp_mem_free, 270
 - fasp_mem_iludata_check, 270
 - fasp_mem_realloc, 271
 - fasp_mem_usage, 271
 - total_alloc_count, 271
 - total_alloc_mem, 271
- message.c, 272
 - fasp_chkerr, 272
 - print_amgcomplexity, 273
 - print_amgcomplexity_bsr, 273
 - print_cputime, 273
 - print_itinfo, 274
 - print_message, 274
- mgcycle.c, 275
 - fasp_solver_mgcycle, 275
 - fasp_solver_mgcycle_bsr, 275
- mgl
 - precond_block_data, 58
- mgl_data
 - precond_FASP_blkoi_data, 70
- mgrecur.c, 276
 - fasp_solver_mgrecur, 276
- Mumps_data, 55
- mxv_matfree, 56
- NEDMALLOC
 - fasp.h, 163
- NL_AMLI_CYCLE
 - fasp_const.h, 180
- NO_ORDER
 - fasp_const.h, 180
- nedmallinfo, 56
- neigh
 - precond_FASP_blkoi_data, 70
- NumLayers
 - precond_sweeping_data, 73
- 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, 70
 - precond_block_reservoir_data, 61
- ordering.c, 277
 - fasp_BinarySearch, 282
 - fasp_aux_dQuickSort, 278
 - fasp_aux_dQuickSortIndex, 278
 - fasp_aux_iQuickSort, 278
 - fasp_aux_iQuickSortIndex, 280
 - fasp_aux_merge, 280
 - fasp_aux_msort, 281
 - fasp_aux_unique, 281
 - fasp_dcsr_CMK_order, 282
 - fasp_dcsr_RCMK_order, 283
- output_type
 - input_param, 48
- p
 - grid2d, 38
- PAIRWISE
 - fasp_const.h, 181
- PP
 - precond_FASP_blkoi_data, 71
 - precond_block_reservoir_data, 61
- 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, 283
 - fasp_param_Schwarz_init, 288
 - fasp_param_Schwarz_print, 290
 - fasp_param_Schwarz_set, 290
 - fasp_param_amg_init, 284
 - fasp_param_amg_print, 285
 - fasp_param_amg_set, 285
 - fasp_param_amg_to_prec, 285
 - fasp_param_amg_to_prec_bsr, 286
 - fasp_param_ilu_init, 286
 - fasp_param_ilu_print, 286
 - fasp_param_ilu_set, 287
 - fasp_param_init, 287
 - fasp_param_input_init, 287
 - fasp_param_prec_to_amg, 288
 - fasp_param_prec_to_amg_bsr, 288
 - fasp_param_set, 290
 - fasp_param_solver_init, 291
 - fasp_param_solver_print, 291
 - fasp_param_solver_set, 291
- Pardiso_data, 56
- pbcgs.c, 292
 - fasp_solver_bdcsr_pbcgs, 293
 - fasp_solver_dbsr_pbcgs, 294
 - fasp_solver_dcsr_pbcgs, 295
 - fasp_solver_dstr_pbcgs, 295
- pbcgs_mf.c, 296
 - fasp_solver_pbcgs, 297
- pcg.c, 298
 - fasp_solver_bdcsr_pcg, 300
 - fasp_solver_dbsr_pcg, 301
 - fasp_solver_dcsr_pcg, 301
 - fasp_solver_dstr_pcg, 303
- pcg_mf.c, 304
 - fasp_solver_pcg, 305
- pcgrid2d
 - fasp.h, 165
- pdiri
 - grid2d, 38
- perf_idx
 - precond_FASP_blkoi_data, 70
 - precond_block_reservoir_data, 61
- perf_neigh
 - precond_FASP_blkoi_data, 71
- pfather
 - grid2d, 38
- pgcg.c, 306
 - fasp_solver_dcsr_pgcg, 306
- pgcg_mf.c, 307
 - fasp_solver_pgcg, 307
- pgcr.c, 308
 - fasp_solver_dcsr_pgcr, 309
 - fasp_solver_dcsr_pgcr1, 310

- pgmres.c, [311](#)
 - fasp_solver_bdcsr_pgmres, [311](#)
 - fasp_solver_dbsr_pgmres, [312](#)
 - fasp_solver_dcsr_pgmres, [313](#)
 - fasp_solver_dstr_pgmres, [313](#)
- pgmres_mf.c, [314](#)
 - fasp_solver_pgmres, [315](#)
- pgrid2d
 - fasp.h, [165](#)
- pivot
 - precond_FASP_blkoi_data, [71](#)
 - precond_block_reservoir_data, [61](#)
- pivot_S
 - precond_FASP_blkoi_data, [71](#)
- pivotS
 - precond_block_reservoir_data, [61](#)
- pminres.c, [315](#)
 - fasp_solver_bdcsr_pminres, [317](#)
 - fasp_solver_dcsr_pminres, [317](#)
 - fasp_solver_dstr_pminres, [318](#)
- pminres_mf.c, [319](#)
 - fasp_solver_pminres, [320](#)
- precond, [57](#)
- precond_FASP_blkoi_data, [68](#)
 - A, [70](#)
 - diaginv, [70](#)
 - diaginv_S, [70](#)
 - diaginv_noscale, [70](#)
 - maxit, [70](#)
 - mgl_data, [70](#)
 - neigh, [70](#)
 - order, [70](#)
 - PP, [71](#)
 - perf_idx, [70](#)
 - perf_neigh, [71](#)
 - pivot, [71](#)
 - pivot_S, [71](#)
 - r, [71](#)
 - RR, [71](#)
 - restart, [71](#)
 - SS, [72](#)
 - scaled, [71](#)
 - tol, [72](#)
 - w, [72](#)
 - WW, [72](#)
- precond_bcsr.c, [321](#)
 - fasp_precond_block_SGS_3, [325](#)
 - fasp_precond_block_SGS_3_amg, [327](#)
 - fasp_precond_block_diag_3, [322](#)
 - fasp_precond_block_diag_3_amg, [323](#)
 - fasp_precond_block_diag_4, [323](#)
 - fasp_precond_block_lower_3, [323](#)
 - fasp_precond_block_lower_3_amg, [325](#)
 - fasp_precond_block_lower_4, [325](#)
 - fasp_precond_block_upper_3, [327](#)
 - fasp_precond_block_upper_3_amg, [327](#)
 - fasp_precond_sweeping, [329](#)
- precond_block_data, [57](#)
 - A_diag, [58](#)
 - Abcsr, [58](#)
 - amgparam, [58](#)
 - LU_diag, [58](#)
 - mgl, [58](#)
 - r, [58](#)
- precond_block_reservoir_data, [59](#)
 - diag, [60](#)
 - diaginv, [60](#)
 - diaginvS, [61](#)
 - fasp_block.h, [169](#)
 - order, [61](#)
 - PP, [61](#)
 - perf_idx, [61](#)
 - pivot, [61](#)
 - pivotS, [61](#)
 - r, [61](#)
 - RR, [61](#)
 - SS, [62](#)
 - scaled, [61](#)
 - w, [62](#)
 - WW, [62](#)
- precond_bsr.c, [329](#)
 - fasp_precond_dbsr_amg, [330](#)
 - fasp_precond_dbsr_amg_nk, [330](#)
 - fasp_precond_dbsr_diag, [331](#)
 - fasp_precond_dbsr_diag_nc2, [331](#)
 - fasp_precond_dbsr_diag_nc3, [332](#)
 - fasp_precond_dbsr_diag_nc5, [332](#)
 - fasp_precond_dbsr_diag_nc7, [334](#)
 - fasp_precond_dbsr_ilu, [334](#)
 - fasp_precond_dbsr_nl_amli, [336](#)
- precond_csr.c, [336](#)
 - fasp_precond_Schwarz, [341](#)
 - fasp_precond_amg, [337](#)
 - fasp_precond_amg_nk, [338](#)
 - fasp_precond_amli, [338](#)
 - fasp_precond_diag, [338](#)
 - fasp_precond_famg, [339](#)
 - fasp_precond_free, [339](#)
 - fasp_precond_ilu, [339](#)
 - fasp_precond_ilu_backward, [340](#)
 - fasp_precond_ilu_forward, [340](#)
 - fasp_precond_nl_amli, [341](#)
 - fasp_precond_setup, [341](#)
- precond_data, [62](#)
- precond_data_bsr, [64](#)
- precond_data_str, [65](#)
- precond_diagbsr, [67](#)
- precond_diagstr, [67](#)

- precond_str.c, [342](#)
 - fasp_precond_dstr_blockgs, [343](#)
 - fasp_precond_dstr_diag, [343](#)
 - fasp_precond_dstr_ilu0, [343](#)
 - fasp_precond_dstr_ilu0_backward, [344](#)
 - fasp_precond_dstr_ilu0_forward, [344](#)
 - fasp_precond_dstr_ilu1, [344](#)
 - fasp_precond_dstr_ilu1_backward, [346](#)
 - fasp_precond_dstr_ilu1_forward, [346](#)
- precond_sweeping_data, [72](#)
 - A, [73](#)
 - Ai, [73](#)
 - local_A, [73](#)
 - local_LU, [73](#)
 - local_index, [73](#)
 - NumLayers, [73](#)
 - r, [74](#)
 - w, [74](#)
- precond_type
 - input_param, [48](#)
 - itsolver_param, [50](#)
- print_amgcomplexity
 - message.c, [273](#)
- print_amgcomplexity_bsr
 - message.c, [273](#)
- print_cputime
 - message.c, [273](#)
- print_itinfo
 - message.c, [274](#)
- print_level
 - input_param, [48](#)
 - itsolver_param, [50](#)
- print_message
 - message.c, [274](#)
- problem_num
 - input_param, [48](#)
- pvfgmres.c, [347](#)
 - fasp_solver_bdcsr_pvfgmres, [347](#)
 - fasp_solver_dbsr_pvfgmres, [348](#)
 - fasp_solver_dcsr_pvfgmres, [349](#)
- pvfgmres_mf.c, [349](#)
 - fasp_solver_pvfgmres, [350](#)
- pvgmres.c, [351](#)
 - fasp_solver_bdcsr_pvgmres, [351](#)
 - fasp_solver_dbsr_pvgmres, [352](#)
 - fasp_solver_dcsr_pvgmres, [352](#)
 - fasp_solver_dstr_pvgmres, [354](#)
- pvgmres_mf.c, [355](#)
 - fasp_solver_pvgmres, [355](#)
- quadrature.c, [356](#)
 - fasp_gauss2d, [356](#)
 - fasp_quad2d, [357](#)
- r
 - precond_FASP_blkcoil_data, [71](#)
 - precond_block_data, [58](#)
 - precond_block_reservoir_data, [61](#)
 - precond_sweeping_data, [74](#)
- REAL
 - fasp.h, [163](#)
- RR
 - precond_FASP_blkcoil_data, [71](#)
 - precond_block_reservoir_data, [61](#)
- RS_C1
 - fasp.h, [163](#)
- rap.c, [357](#)
 - fasp_blas_dcsr_rap2, [358](#)
- restart
 - input_param, [48](#)
 - itsolver_param, [50](#)
 - precond_FASP_blkcoil_data, [71](#)
- 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_BLKCOIL
 - 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_PARDISO
 fasp_const.h, 186
 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, 187
 SOLVER_UMFPACK
 fasp_const.h, 187
 SOLVER_VFGMRES
 fasp_const.h, 187
 SOLVER_VGMRES
 fasp_const.h, 187
 SS
 precond_FASP_blkoi_data, 72
 precond_block_reservoir_data, 62
 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, 362
 scaled
 precond_FASP_blkoi_data, 71
 precond_block_reservoir_data, 61
 Schwarz_blk solver
 input_param, 48
 Schwarz_data, 74
 Schwarz_maxlvl
 input_param, 48
 Schwarz_mmsize
 input_param, 49
 Schwarz_param, 75
 schwarz_setup.c, 358
 fasp_Schwarz_get_block_matrix, 360
 fasp_Schwarz_setup, 360
 fasp_dcsr_Schwarz_backward_smoother, 359
 fasp_dcsr_Schwarz_forward_smoother, 359
 Schwarz_type
 input_param, 49
 smat.c, 361
 fasp_blas_smat_Linfinity, 364
 fasp_blas_smat_inv, 362
 fasp_blas_smat_inv_nc, 362
 fasp_blas_smat_inv_nc2, 362
 fasp_blas_smat_inv_nc3, 363
 fasp_blas_smat_inv_nc4, 363
 fasp_blas_smat_inv_nc5, 363
 fasp_blas_smat_inv_nc7, 364
 fasp_blas_smat_invp_nc, 364
 fasp_iden_free, 365
 fasp_smat_identity, 365
 fasp_smat_identity_nc2, 365
 fasp_smat_identity_nc3, 367
 fasp_smat_identity_nc5, 367
 fasp_smat_identity_nc7, 367
 SWAP, 362
 smoother_bsr.c, 368
 fasp_smoother_dbsr_gs, 369
 fasp_smoother_dbsr_gs1, 369
 fasp_smoother_dbsr_gs_ascend, 370
 fasp_smoother_dbsr_gs_ascend1, 370
 fasp_smoother_dbsr_gs_descend, 371
 fasp_smoother_dbsr_gs_descend1, 371
 fasp_smoother_dbsr_gs_order1, 372
 fasp_smoother_dbsr_gs_order2, 372
 fasp_smoother_dbsr_ilu, 373
 fasp_smoother_dbsr_jacobi, 373
 fasp_smoother_dbsr_jacobi1, 373

- fasp_smoother_dbsr_jacobi_setup, 375
 - fasp_smoother_dbsr_sor, 375
 - fasp_smoother_dbsr_sor1, 376
 - fasp_smoother_dbsr_sor_ascend, 376
 - fasp_smoother_dbsr_sor_descend, 377
 - fasp_smoother_dbsr_sor_order, 377
- smoother_csr.c, 378
 - fasp_smoother_dcsr_L1diag, 382
 - fasp_smoother_dcsr_gs, 379
 - fasp_smoother_dcsr_gs_cf, 379
 - fasp_smoother_dcsr_gs_rb3d, 380
 - fasp_smoother_dcsr_ilu, 380
 - fasp_smoother_dcsr_jacobi, 381
 - fasp_smoother_dcsr_kaczmarz, 381
 - fasp_smoother_dcsr_sgs, 382
 - fasp_smoother_dcsr_sor, 383
 - fasp_smoother_dcsr_sor_cf, 383
- smoother_csr_cr.c, 384
 - fasp_smoother_dcsr_gscr, 384
- smoother_csr_poly.c, 385
 - fasp_smoother_dcsr_poly, 386
 - fasp_smoother_dcsr_poly_old, 386
- smoother_str.c, 386
 - fasp_generate_diaginv_block, 388
 - fasp_smoother_dstr_gs, 388
 - fasp_smoother_dstr_gs1, 388
 - fasp_smoother_dstr_gs_ascend, 389
 - fasp_smoother_dstr_gs_cf, 389
 - fasp_smoother_dstr_gs_descend, 390
 - fasp_smoother_dstr_gs_order, 390
 - fasp_smoother_dstr_jacobi, 391
 - fasp_smoother_dstr_jacobi1, 391
 - fasp_smoother_dstr_schwarz, 392
 - fasp_smoother_dstr_sor, 392
 - fasp_smoother_dstr_sor1, 392
 - fasp_smoother_dstr_sor_ascend, 393
 - fasp_smoother_dstr_sor_cf, 393
 - fasp_smoother_dstr_sor_descend, 394
 - fasp_smoother_dstr_sor_order, 394
- solver_type
 - input_param, 49
- sparse_block.c, 395
 - fasp_bdcsr_free, 396
 - fasp_dbsr_Linfinity_dcsr, 398
 - fasp_dbsr_getblk, 397
 - fasp_dbsr_getblk_dcsr, 397
 - fasp_dcsr_getblk, 398
- sparse_bsr.c, 399
 - fasp_dbsr_alloc, 400
 - fasp_dbsr_cp, 400
 - fasp_dbsr_create, 401
 - fasp_dbsr_diagLU, 403
 - fasp_dbsr_diagLU2, 404
 - fasp_dbsr_diaginv, 401
 - fasp_dbsr_diaginv2, 402
 - fasp_dbsr_diaginv3, 402
 - fasp_dbsr_diaginv4, 403
 - fasp_dbsr_diagpref, 404
 - fasp_dbsr_free, 405
 - fasp_dbsr_getdiag, 405
 - fasp_dbsr_getdiaginv, 406
 - fasp_dbsr_null, 406
 - fasp_dbsr_trans, 406
- sparse_coo.c, 407
 - fasp_dcoo_alloc, 407
 - fasp_dcoo_create, 408
 - fasp_dcoo_free, 408
 - fasp_dcoo_shift, 408
- sparse_csr.c, 409
 - fasp_dcsr_alloc, 410
 - fasp_dcsr_compress, 411
 - fasp_dcsr_compress_inplace, 411
 - fasp_dcsr_cp, 412
 - fasp_dcsr_create, 412
 - fasp_dcsr_diagpref, 412
 - fasp_dcsr_free, 414
 - fasp_dcsr_getcol, 414
 - fasp_dcsr_getdiag, 415
 - fasp_dcsr_multicoloring, 415
 - fasp_dcsr_null, 416
 - fasp_dcsr_perm, 416
 - fasp_dcsr_permz, 416
 - fasp_dcsr_regdiag, 417
 - fasp_dcsr_shift, 417
 - fasp_dcsr_sort, 418
 - fasp_dcsr_sortz, 418
 - fasp_dcsr_symdiagscale, 418
 - fasp_dcsr_sympat, 419
 - fasp_dcsr_trans, 419
 - fasp_dcsr_transz, 420
 - fasp_icsr_cp, 420
 - fasp_icsr_create, 421
 - fasp_icsr_free, 421
 - fasp_icsr_null, 421
 - fasp_icsr_trans, 423
- sparse_csrl.c, 423
 - fasp_dcsrl_create, 424
 - fasp_dcsrl_free, 424
- sparse_str.c, 424
 - fasp_dstr_alloc, 425
 - fasp_dstr_cp, 425
 - fasp_dstr_create, 427
 - fasp_dstr_free, 427
 - fasp_dstr_null, 428
- sparse_util.c, 428
 - fasp_sparse_MIS, 432
 - fasp_sparse_aat, 429
 - fasp_sparse_abyb, 430

- fasp_sparse_abybms_, 430
 - fasp_sparse_aplbms_, 431
 - fasp_sparse_aplusb_, 431
 - fasp_sparse_iit_, 431
 - fasp_sparse_rapcmp_, 432
 - fasp_sparse_rapms_, 433
 - fasp_sparse_wta_, 433
 - fasp_sparse_wtams_, 434
 - fasp_sparse_ytx_, 434
 - fasp_sparse_ytxbig_, 435
- spbcgs.c, 435
 - fasp_solver_bdcsr_spbcgs, 437
 - fasp_solver_dbsr_spbcgs, 437
 - fasp_solver_dcsr_spbcgs, 438
 - fasp_solver_dstr_spbcgs, 438
- spcg.c, 440
 - fasp_solver_bdcsr_spcg, 442
 - fasp_solver_dcsr_spcg, 442
 - fasp_solver_dstr_spcg, 443
- spgmres.c, 444
 - fasp_solver_bdcsr_spgmres, 444
 - fasp_solver_dbsr_spgmres, 445
 - fasp_solver_dcsr_spgmres, 445
 - fasp_solver_dstr_spgmres, 447
- spminres.c, 448
 - fasp_solver_bdcsr_spminres, 449
 - fasp_solver_dcsr_spminres, 450
 - fasp_solver_dstr_spminres, 450
- spvgmres.c, 452
 - fasp_solver_bdcsr_spvgmres, 453
 - fasp_solver_dbsr_spvgmres, 453
 - fasp_solver_dcsr_spvgmres, 454
 - fasp_solver_dstr_spvgmres, 454
- stop_type
 - input_param, 49
 - itsolver_param, 50
- t
 - grid2d, 38
- THDs_AMG_GS
 - threads.c, 456
- THDs_CPR_gGS
 - threads.c, 456
- THDs_CPR_IGS
 - threads.c, 457
- TRUE
 - fasp_const.h, 188
- tfather
 - grid2d, 38
- threads.c, 455
 - FASP_GET_START_END, 456
 - fasp_set_GS_threads, 456
 - THDs_AMG_GS, 456
 - THDs_CPR_gGS, 456
 - THDs_CPR_IGS, 457
 - THDs_CPR_IGS, 457
 - timing.c, 457
 - fasp_gettime, 457
 - tol
 - itsolver_param, 50
 - precond_FASP_blkoi_data, 72
 - total_alloc_count
 - fasp.h, 166
 - memory.c, 271
 - total_alloc_mem
 - fasp.h, 166
 - memory.c, 271
 - 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, 458
 - fasp_dvec_alloc, 459
 - fasp_dvec_cp, 460
 - fasp_dvec_create, 460
 - fasp_dvec_free, 460
 - fasp_dvec_isnan, 462
 - fasp_dvec_maxdiff, 462
 - fasp_dvec_null, 463
 - fasp_dvec_rand, 463
 - fasp_dvec_set, 464
 - fasp_dvec_symdiagscale, 464
 - fasp_ivec_alloc, 464
 - fasp_ivec_create, 465
 - fasp_ivec_free, 465
 - fasp_ivec_set, 465
 - vertices
 - grid2d, 39
 - w
 - precond_FASP_blkoi_data, 72
 - precond_block_reservoir_data, 62
 - precond_sweeping_data, 74
 - W_CYCLE
 - fasp_const.h, 188
 - WW
 - precond_FASP_blkoi_data, 72
 - precond_block_reservoir_data, 62

workdir

input_param, [49](#)

wrapper.c, [466](#)

 fasp_fwrapper_amg_, [467](#)

 fasp_fwrapper_krylov_amg_, [467](#)

 fasp_wrapper_dbsr_krylov_amg, [468](#)

 fasp_wrapper_dcoo_dbsr_krylov_amg, [468](#)