

Fast Auxiliary Space Preconditioning

1.7.7 Sep/04/2015

Generated by Doxygen 1.8.10

Fri Sep 4 2015 17:39:56

Contents

1	Introduction	1
2	How to obtain FASP	3
3	Building and Installation	5
4	Developers	7
5	Doxygen	9
6	Data Structure Index	11
6.1	Data Structures	11
7	File Index	13
7.1	File List	13
8	Data Structure Documentation	19
8.1	AMG_data Struct Reference	19
8.1.1	Detailed Description	20
8.2	AMG_data_bsr Struct Reference	20
8.2.1	Detailed Description	21
8.3	AMG_param Struct Reference	22
8.3.1	Detailed Description	24
8.4	block_BSR Struct Reference	24
8.4.1	Detailed Description	24
8.5	block_dCSRmat Struct Reference	24
8.5.1	Detailed Description	25
8.6	block_dvector Struct Reference	25
8.6.1	Detailed Description	25
8.7	block_iCSRmat Struct Reference	25
8.7.1	Detailed Description	26

8.8	block_ivector Struct Reference	26
8.8.1	Detailed Description	26
8.9	block_Reservoir Struct Reference	27
8.9.1	Detailed Description	27
8.10	dBSRmat Struct Reference	27
8.10.1	Detailed Description	28
8.10.2	Field Documentation	28
8.10.2.1	JA	28
8.10.2.2	val	28
8.11	dCOOmat Struct Reference	28
8.11.1	Detailed Description	29
8.12	dCSRLmat Struct Reference	29
8.12.1	Detailed Description	30
8.13	dCSRmat Struct Reference	30
8.13.1	Detailed Description	30
8.14	ddenmat Struct Reference	30
8.14.1	Detailed Description	31
8.15	dSTRmat Struct Reference	31
8.15.1	Detailed Description	32
8.16	dvector Struct Reference	32
8.16.1	Detailed Description	32
8.17	grid2d Struct Reference	32
8.17.1	Detailed Description	33
8.17.2	Field Documentation	33
8.17.2.1	e	33
8.17.2.2	edges	33
8.17.2.3	ediri	33
8.17.2.4	efather	33
8.17.2.5	p	33
8.17.2.6	pdiri	34
8.17.2.7	pfather	34
8.17.2.8	s	34
8.17.2.9	t	34
8.17.2.10	tfather	34
8.17.2.11	triangles	34
8.17.2.12	vertices	34
8.18	iCOOmat Struct Reference	34

8.18.1 Detailed Description	35
8.19 iCSRmat Struct Reference	35
8.19.1 Detailed Description	36
8.20 idenmat Struct Reference	36
8.20.1 Detailed Description	36
8.21 ILU_data Struct Reference	36
8.21.1 Detailed Description	37
8.22 ILU_param Struct Reference	37
8.22.1 Detailed Description	38
8.23 input_param Struct Reference	38
8.23.1 Detailed Description	39
8.23.2 Field Documentation	39
8.23.2.1 AMG_aggregation_type	39
8.23.2.2 AMG_aggressive_level	39
8.23.2.3 AMG_aggressive_path	39
8.23.2.4 AMG_amli_degree	39
8.23.2.5 AMG_coarse_dof	40
8.23.2.6 AMG_coarse_scaling	40
8.23.2.7 AMG_coarse_solver	40
8.23.2.8 AMG_coarsening_type	40
8.23.2.9 AMG_cycle_type	40
8.23.2.10 AMG_ILU_levels	40
8.23.2.11 AMG_interpolation_type	40
8.23.2.12 AMG_levels	40
8.23.2.13 AMG_max_aggregation	40
8.23.2.14 AMG_max_row_sum	41
8.23.2.15 AMG_maxit	41
8.23.2.16 AMG_nl_amli_krylov_type	41
8.23.2.17 AMG_pair_number	41
8.23.2.18 AMG_polynomial_degree	41
8.23.2.19 AMG_postsmooth_iter	41
8.23.2.20 AMG_presmooth_iter	41
8.23.2.21 AMG_quality_bound	41
8.23.2.22 AMG_relaxation	41
8.23.2.23 AMG_Schwarz_levels	42
8.23.2.24 AMG_smooth_filter	42
8.23.2.25 AMG_smooth_order	42

8.23.2.26 AMG_smoother	42
8.23.2.27 AMG_strong_coupled	42
8.23.2.28 AMG_strong_threshold	42
8.23.2.29 AMG_tentative_smooth	42
8.23.2.30 AMG_tol	42
8.23.2.31 AMG_truncation_threshold	42
8.23.2.32 AMG_type	43
8.23.2.33 ILU_droptol	43
8.23.2.34 ILU_lfil	43
8.23.2.35 ILU_permtol	43
8.23.2.36 ILU_relax	43
8.23.2.37 ILU_type	43
8.23.2.38 inifile	43
8.23.2.39 itsolver_maxit	43
8.23.2.40 itsolver_tol	43
8.23.2.41 output_type	44
8.23.2.42 precondition_type	44
8.23.2.43 print_level	44
8.23.2.44 problem_num	44
8.23.2.45 restart	44
8.23.2.46 Schwarz_blksolver	44
8.23.2.47 Schwarz_maxlvl	44
8.23.2.48 Schwarz_mmsize	44
8.23.2.49 Schwarz_type	44
8.23.2.50 solver_type	45
8.23.2.51 stop_type	45
8.23.2.52 workdir	45
8.24 itsolver_param Struct Reference	45
8.24.1 Detailed Description	45
8.24.2 Field Documentation	45
8.24.2.1 itsolver_type	45
8.24.2.2 maxit	46
8.24.2.3 precondition_type	46
8.24.2.4 print_level	46
8.24.2.5 restart	46
8.24.2.6 stop_type	46
8.24.2.7 tol	46

8.25	ivector Struct Reference	46
8.25.1	Detailed Description	47
8.26	Link Struct Reference	47
8.26.1	Detailed Description	47
8.27	linked_list Struct Reference	47
8.27.1	Detailed Description	48
8.28	Mumps_data Struct Reference	48
8.28.1	Detailed Description	48
8.29	mxv_matfree Struct Reference	48
8.29.1	Detailed Description	49
8.30	precond Struct Reference	49
8.30.1	Detailed Description	49
8.31	precond_block_data Struct Reference	49
8.31.1	Detailed Description	50
8.31.2	Field Documentation	50
8.31.2.1	A_diag	50
8.31.2.2	Abcsr	50
8.31.2.3	amgparam	50
8.31.2.4	LU_diag	50
8.31.2.5	mgl	50
8.31.2.6	r	50
8.32	precond_block_reservoir_data Struct Reference	51
8.32.1	Detailed Description	52
8.32.2	Field Documentation	52
8.32.2.1	diag	52
8.32.2.2	diaginv	52
8.32.2.3	diaginvS	52
8.32.2.4	order	53
8.32.2.5	perf_idx	53
8.32.2.6	pivot	53
8.32.2.7	pivotS	53
8.32.2.8	PP	53
8.32.2.9	r	53
8.32.2.10	RR	53
8.32.2.11	scaled	53
8.32.2.12	SS	53
8.32.2.13	w	54

8.32.2.14 WW	54
8.33 precondition_data Struct Reference	54
8.33.1 Detailed Description	55
8.34 precondition_data_bsr Struct Reference	55
8.34.1 Detailed Description	57
8.35 precondition_data_str Struct Reference	57
8.35.1 Detailed Description	58
8.36 precondition_diagbsr Struct Reference	59
8.36.1 Detailed Description	59
8.37 precondition_diagstr Struct Reference	59
8.37.1 Detailed Description	59
8.38 precondition_FASP_blkoi_data Struct Reference	60
8.38.1 Detailed Description	61
8.38.2 Field Documentation	61
8.38.2.1 A	61
8.38.2.2 diaginv	61
8.38.2.3 diaginv_noscale	62
8.38.2.4 diaginv_S	62
8.38.2.5 maxit	62
8.38.2.6 mgl_data	62
8.38.2.7 neigh	62
8.38.2.8 order	62
8.38.2.9 perf_idx	62
8.38.2.10 perf_neigh	62
8.38.2.11 pivot	62
8.38.2.12 pivot_S	63
8.38.2.13 PP	63
8.38.2.14 r	63
8.38.2.15 restart	63
8.38.2.16 RR	63
8.38.2.17 scaled	63
8.38.2.18 SS	63
8.38.2.19 tol	63
8.38.2.20 w	64
8.38.2.21 WW	64
8.39 precondition_sweeping_data Struct Reference	64
8.39.1 Detailed Description	64

8.39.2	Field Documentation	65
8.39.2.1	A	65
8.39.2.2	Ai	65
8.39.2.3	local_A	65
8.39.2.4	local_index	65
8.39.2.5	local_LU	65
8.39.2.6	NumLayers	65
8.39.2.7	r	65
8.39.2.8	w	65
8.40	Schwarz_data Struct Reference	66
8.40.1	Detailed Description	67
8.41	Schwarz_param Struct Reference	67
8.41.1	Detailed Description	67
9	File Documentation	69
9.1	amg.c File Reference	69
9.1.1	Detailed Description	69
9.1.2	Function Documentation	69
9.1.2.1	fasp_solver_amg(dCSRmat *A, dvector *b, dvector *x, AMG_param *param)	69
9.2	amg_setup_cr.c File Reference	70
9.2.1	Detailed Description	70
9.2.2	Function Documentation	70
9.2.2.1	fasp_amg_setup_cr(AMG_data *mgl, AMG_param *param)	70
9.3	amg_setup_rs.c File Reference	71
9.3.1	Detailed Description	71
9.3.2	Function Documentation	71
9.3.2.1	fasp_amg_setup_rs(AMG_data *mgl, AMG_param *param)	71
9.4	amg_setup_sa.c File Reference	72
9.4.1	Detailed Description	72
9.4.2	Function Documentation	73
9.4.2.1	fasp_amg_setup_sa(AMG_data *mgl, AMG_param *param)	73
9.4.2.2	fasp_amg_setup_sa_bsr(AMG_data_bsr *mgl, AMG_param *param)	73
9.5	amg_setup_ua.c File Reference	74
9.5.1	Detailed Description	74
9.5.2	Function Documentation	74
9.5.2.1	fasp_amg_setup_ua(AMG_data *mgl, AMG_param *param)	74
9.5.2.2	fasp_amg_setup_ua_bsr(AMG_data_bsr *mgl, AMG_param *param)	75

9.6	amg_solve.c File Reference	76
9.6.1	Detailed Description	76
9.6.2	Function Documentation	77
9.6.2.1	fasp_amg_solve(AMG_data *mgl, AMG_param *param)	77
9.6.2.2	fasp_amg_solve_amli(AMG_data *mgl, AMG_param *param)	78
9.6.2.3	fasp_amg_solve_nl_amli(AMG_data *mgl, AMG_param *param)	78
9.6.2.4	fasp_famg_solve(AMG_data *mgl, AMG_param *param)	79
9.7	amlirecur.c File Reference	79
9.7.1	Detailed Description	80
9.7.2	Function Documentation	80
9.7.2.1	fasp_amg_amli_coef(const REAL lambda_max, const REAL lambda_min, const INT degree, REAL *coef)	80
9.7.2.2	fasp_solver_amli(AMG_data *mgl, AMG_param *param, INT level)	80
9.7.2.3	fasp_solver_nl_amli(AMG_data *mgl, AMG_param *param, INT level, INT num_levels)	81
9.7.2.4	fasp_solver_nl_amli_bsr(AMG_data_bsr *mgl, AMG_param *param, INT level, INT num_levels)	81
9.8	array.c File Reference	82
9.8.1	Detailed Description	83
9.8.2	Function Documentation	83
9.8.2.1	fasp_array_cp(const INT n, REAL *x, REAL *y)	83
9.8.2.2	fasp_array_cp_nc3(REAL *x, REAL *y)	83
9.8.2.3	fasp_array_cp_nc5(REAL *x, REAL *y)	83
9.8.2.4	fasp_array_cp_nc7(REAL *x, REAL *y)	84
9.8.2.5	fasp_array_null(REAL *x)	84
9.8.2.6	fasp_array_set(const INT n, REAL *x, const REAL val)	85
9.8.2.7	fasp_ialray_cp(const INT n, INT *x, INT *y)	85
9.8.2.8	fasp_ialray_set(const INT n, INT *x, const INT val)	85
9.9	blas_array.c File Reference	86
9.9.1	Detailed Description	87
9.9.2	Function Documentation	87
9.9.2.1	fasp_blas_array_ax(const INT n, const REAL a, REAL *x)	87
9.9.2.2	fasp_blas_array_axpby(const INT n, const REAL a, REAL *x, const REAL b, REAL *y)	87
9.9.2.3	fasp_blas_array_axpy(const INT n, const REAL a, REAL *x, REAL *y)	88
9.9.2.4	fasp_blas_array_axpyz(const INT n, const REAL a, REAL *x, REAL *y, REAL *z)	88
9.9.2.5	fasp_blas_array_dotprod(const INT n, const REAL *x, const REAL *y)	89
9.9.2.6	fasp_blas_array_norm1(const INT n, const REAL *x)	90
9.9.2.7	fasp_blas_array_norm2(const INT n, const REAL *x)	90

9.9.2.8	<code>fasp_blas_array_norminf(const INT n, const REAL *x)</code>	91
9.10	<code>blas_bcsr.c</code> File Reference	91
9.10.1	Detailed Description	92
9.10.2	Function Documentation	92
9.10.2.1	<code>fasp_blas_bdbsr_aApy(const REAL alpha, block_BSR *A, REAL *x, REAL *y)</code>	92
9.10.2.2	<code>fasp_blas_bdbsr_mxv(block_BSR *A, REAL *x, REAL *y)</code>	92
9.10.2.3	<code>fasp_blas_bdcsr_aApy(const REAL alpha, block_dCSRmat *A, REAL *x, REAL *y)</code>	93
9.10.2.4	<code>fasp_blas_bdcsr_mxv(block_dCSRmat *A, REAL *x, REAL *y)</code>	93
9.11	<code>blas_bsr.c</code> File Reference	93
9.11.1	Detailed Description	94
9.11.2	Function Documentation	94
9.11.2.1	<code>fasp_blas_dbsr_aApyby(const REAL alpha, dBSRmat *A, REAL *x, const REAL beta, REAL *y)</code>	94
9.11.2.2	<code>fasp_blas_dbsr_aApy(const REAL alpha, dBSRmat *A, REAL *x, REAL *y)</code>	95
9.11.2.3	<code>fasp_blas_dbsr_aApy_agg(const REAL alpha, dBSRmat *A, REAL *x, REAL *y)</code>	95
9.11.2.4	<code>fasp_blas_dbsr_axm(dBSRmat *A, const REAL alpha)</code>	96
9.11.2.5	<code>fasp_blas_dbsr_mxm(dBSRmat *A, dBSRmat *B, dBSRmat *C)</code>	96
9.11.2.6	<code>fasp_blas_dbsr_mxv(dBSRmat *A, REAL *x, REAL *y)</code>	97
9.11.2.7	<code>fasp_blas_dbsr_mxv_agg(dBSRmat *A, REAL *x, REAL *y)</code>	97
9.11.2.8	<code>fasp_blas_dbsr_rap(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	98
9.11.2.9	<code>fasp_blas_dbsr_rap1(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	99
9.11.2.10	<code>fasp_blas_dbsr_rap_agg(dBSRmat *R, dBSRmat *A, dBSRmat *P, dBSRmat *B)</code>	99
9.12	<code>blas_csr.c</code> File Reference	100
9.12.1	Detailed Description	101
9.12.2	Function Documentation	101
9.12.2.1	<code>fasp_blas_dcsr_aApy(const REAL alpha, dCSRmat *A, REAL *x, REAL *y)</code>	101
9.12.2.2	<code>fasp_blas_dcsr_aApy_agg(const REAL alpha, dCSRmat *A, REAL *x, REAL *y)</code>	102
9.12.2.3	<code>fasp_blas_dcsr_add(dCSRmat *A, const REAL alpha, dCSRmat *B, const REAL beta, dCSRmat *C)</code>	103
9.12.2.4	<code>fasp_blas_dcsr_axm(dCSRmat *A, const REAL alpha)</code>	103
9.12.2.5	<code>fasp_blas_dcsr_bandwith(dCSRmat *A, INT *bndwith)</code>	104
9.12.2.6	<code>fasp_blas_dcsr_mxm(dCSRmat *A, dCSRmat *B, dCSRmat *C)</code>	104
9.12.2.7	<code>fasp_blas_dcsr_mxv(dCSRmat *A, REAL *x, REAL *y)</code>	105
9.12.2.8	<code>fasp_blas_dcsr_mxv_agg(dCSRmat *A, REAL *x, REAL *y)</code>	105
9.12.2.9	<code>fasp_blas_dcsr_ptap(dCSRmat *Pt, dCSRmat *A, dCSRmat *P, dCSRmat *Ac)</code>	105
9.12.2.10	<code>fasp_blas_dcsr_rap(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *RAP)</code>	106
9.12.2.11	<code>fasp_blas_dcsr_rap4(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *B, INT *icor_ysk)</code>	106

9.12.2.12 fasp_blas_dcsr_rap_agg(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *RAP)	107
9.12.2.13 fasp_blas_dcsr_rap_agg1(dCSRmat *R, dCSRmat *A, dCSRmat *P, dCSRmat *B)	107
9.12.2.14 fasp_blas_dcsr_vmv(dCSRmat *A, REAL *x, REAL *y)	108
9.13 blas_csrl.c File Reference	108
9.13.1 Detailed Description	109
9.13.2 Function Documentation	109
9.13.2.1 fasp_blas_dcsrl_mnv(dCSRLmat *A, REAL *x, REAL *y)	109
9.14 blas_smat.c File Reference	109
9.14.1 Detailed Description	111
9.14.2 Function Documentation	111
9.14.2.1 fasp_blas_array_axpy_nc2(const REAL a, REAL *x, REAL *y)	111
9.14.2.2 fasp_blas_array_axpy_nc3(const REAL a, REAL *x, REAL *y)	111
9.14.2.3 fasp_blas_array_axpy_nc5(const REAL a, REAL *x, REAL *y)	112
9.14.2.4 fasp_blas_array_axpy_nc7(const REAL a, REAL *x, REAL *y)	112
9.14.2.5 fasp_blas_array_axpyz_nc2(REAL a, REAL *x, REAL *y, REAL *z)	112
9.14.2.6 fasp_blas_array_axpyz_nc3(const REAL a, REAL *x, REAL *y, REAL *z)	113
9.14.2.7 fasp_blas_array_axpyz_nc5(const REAL a, REAL *x, REAL *y, REAL *z)	113
9.14.2.8 fasp_blas_array_axpyz_nc7(const REAL a, REAL *x, REAL *y, REAL *z)	114
9.14.2.9 fasp_blas_smat_aAxpby(const REAL alpha, REAL *A, REAL *x, const REAL beta, REAL *y, const INT n)	114
9.14.2.10 fasp_blas_smat_add(REAL *a, REAL *b, const INT n, const REAL alpha, const REAL beta, REAL *c)	115
9.14.2.11 fasp_blas_smat_axm(REAL *a, const INT n, const REAL alpha)	115
9.14.2.12 fasp_blas_smat_mul(REAL *a, REAL *b, REAL *c, const INT n)	116
9.14.2.13 fasp_blas_smat_mul_nc2(REAL *a, REAL *b, REAL *c)	116
9.14.2.14 fasp_blas_smat_mul_nc3(REAL *a, REAL *b, REAL *c)	117
9.14.2.15 fasp_blas_smat_mul_nc5(REAL *a, REAL *b, REAL *c)	118
9.14.2.16 fasp_blas_smat_mul_nc7(REAL *a, REAL *b, REAL *c)	118
9.14.2.17 fasp_blas_smat_mnv(REAL *a, REAL *b, REAL *c, const INT n)	119
9.14.2.18 fasp_blas_smat_mnv_nc2(REAL *a, REAL *b, REAL *c)	120
9.14.2.19 fasp_blas_smat_mnv_nc3(REAL *a, REAL *b, REAL *c)	120
9.14.2.20 fasp_blas_smat_mnv_nc5(REAL *a, REAL *b, REAL *c)	121
9.14.2.21 fasp_blas_smat_mnv_nc7(REAL *a, REAL *b, REAL *c)	122
9.14.2.22 fasp_blas_smat_ymAx(REAL *A, REAL *x, REAL *y, INT n)	122
9.14.2.23 fasp_blas_smat_ymAx_nc2(REAL *A, REAL *x, REAL *y)	123
9.14.2.24 fasp_blas_smat_ymAx_nc3(REAL *A, REAL *x, REAL *y)	124
9.14.2.25 fasp_blas_smat_ymAx_nc5(REAL *A, REAL *x, REAL *y)	124

9.14.2.26	<code>fasp_blas_smat_ymAx_nc7(REAL *A, REAL *x, REAL *y)</code>	125
9.14.2.27	<code>fasp_blas_smat_ymAx_ns(REAL *A, REAL *x, REAL *y, const INT n)</code>	125
9.14.2.28	<code>fasp_blas_smat_ymAx_ns2(REAL *A, REAL *x, REAL *y)</code>	126
9.14.2.29	<code>fasp_blas_smat_ymAx_ns3(REAL *A, REAL *x, REAL *y)</code>	126
9.14.2.30	<code>fasp_blas_smat_ymAx_ns5(REAL *A, REAL *x, REAL *y)</code>	127
9.14.2.31	<code>fasp_blas_smat_ymAx_ns7(REAL *A, REAL *x, REAL *y)</code>	127
9.14.2.32	<code>fasp_blas_smat_ypAx(REAL *A, REAL *x, REAL *y, const INT n)</code>	128
9.14.2.33	<code>fasp_blas_smat_ypAx_nc2(REAL *A, REAL *x, REAL *y)</code>	128
9.14.2.34	<code>fasp_blas_smat_ypAx_nc3(REAL *A, REAL *x, REAL *y)</code>	129
9.14.2.35	<code>fasp_blas_smat_ypAx_nc5(REAL *A, REAL *x, REAL *y)</code>	129
9.14.2.36	<code>fasp_blas_smat_ypAx_nc7(REAL *A, REAL *x, REAL *y)</code>	129
9.15	<code>blas_str.c</code> File Reference	130
9.15.1	Detailed Description	130
9.15.2	Function Documentation	130
9.15.2.1	<code>fasp_blas_dstr_aApy(REAL alpha, dSTRmat *A, REAL *x, REAL *y)</code>	130
9.15.2.2	<code>fasp_blas_dstr_mxv(dSTRmat *A, REAL *x, REAL *y)</code>	131
9.15.2.3	<code>fasp_dstr_diagscale(dSTRmat *A, dSTRmat *B)</code>	131
9.16	<code>blas_vec.c</code> File Reference	132
9.16.1	Detailed Description	132
9.16.2	Function Documentation	132
9.16.2.1	<code>fasp_blas_dvec_axpy(const REAL a, dvector *x, dvector *y)</code>	132
9.16.2.2	<code>fasp_blas_dvec_axpyz(const REAL a, dvector *x, dvector *y, dvector *z)</code>	133
9.16.2.3	<code>fasp_blas_dvec_dotprod(dvector *x, dvector *y)</code>	133
9.16.2.4	<code>fasp_blas_dvec_norm1(dvector *x)</code>	134
9.16.2.5	<code>fasp_blas_dvec_norm2(dvector *x)</code>	134
9.16.2.6	<code>fasp_blas_dvec_norminf(dvector *x)</code>	135
9.16.2.7	<code>fasp_blas_dvec_relerr(dvector *x, dvector *y)</code>	135
9.17	<code>checkmat.c</code> File Reference	136
9.17.1	Detailed Description	137
9.17.2	Function Documentation	137
9.17.2.1	<code>fasp_check_dCSRmat(dCSRmat *A)</code>	137
9.17.2.2	<code>fasp_check_diagdom(dCSRmat *A)</code>	137
9.17.2.3	<code>fasp_check_diagpos(dCSRmat *A)</code>	138
9.17.2.4	<code>fasp_check_diagzero(dCSRmat *A)</code>	139
9.17.2.5	<code>fasp_check_iCSRmat(iCSRmat *A)</code>	139
9.17.2.6	<code>fasp_check_symm(dCSRmat *A)</code>	140
9.18	<code>coarsening_cr.c</code> File Reference	141

9.18.1 Detailed Description	141
9.18.2 Function Documentation	141
9.18.2.1 fasp_amg_coarsening_cr(INT i_0, INT i_n, dCSRmat *A, ivector *vertices, AMG_param *param)	141
9.19 coarsening_rs.c File Reference	142
9.19.1 Detailed Description	142
9.19.2 Function Documentation	143
9.19.2.1 fasp_amg_coarsening_rs(dCSRmat *A, ivector *vertices, dCSRmat *P, iCSRmat *S, AMG_param *param)	143
9.20 convert.c File Reference	144
9.20.1 Detailed Description	145
9.20.2 Function Documentation	145
9.20.2.1 endian_convert_int(const INT inum, const INT ilength, const INT endianflag)	145
9.20.2.2 endian_convert_real(const REAL rnum, INT vlength, INT endianflag)	145
9.20.2.3 fasp_aux_bbyteToldouble(unsigned char bytes[])	146
9.20.2.4 fasp_aux_change_endian4(unsigned long x)	147
9.20.2.5 fasp_aux_change_endian8(double x)	147
9.21 doxygen.h File Reference	148
9.21.1 Detailed Description	148
9.22 eigen.c File Reference	148
9.22.1 Detailed Description	148
9.22.2 Function Documentation	148
9.22.2.1 fasp_dcsr_eig(dCSRmat *A, const REAL tol, const INT maxit)	148
9.23 factor.f File Reference	149
9.23.1 Detailed Description	149
9.24 famg.c File Reference	149
9.24.1 Detailed Description	150
9.24.2 Function Documentation	150
9.24.2.1 fasp_solver_famg(dCSRmat *A, dvector *b, dvector *x, AMG_param *param)	150
9.25 fasp.h File Reference	150
9.25.1 Detailed Description	153
9.25.2 Macro Definition Documentation	153
9.25.2.1 __FASP_HEADER__	153
9.25.2.2 ABS	153
9.25.2.3 DIAGONAL_PREF	153
9.25.2.4 DLMALLOC	153
9.25.2.5 FASP_GSRB	154

9.25.2.6	FASP_USE_ILU	154
9.25.2.7	GE	154
9.25.2.8	GT	154
9.25.2.9	INT	154
9.25.2.10	ISNAN	154
9.25.2.11	LE	154
9.25.2.12	LONG	154
9.25.2.13	LONGLONG	155
9.25.2.14	LS	155
9.25.2.15	MAX	155
9.25.2.16	MIN	155
9.25.2.17	NEDMALLOC	155
9.25.2.18	PUT_INT	155
9.25.2.19	PUT_REAL	155
9.25.2.20	REAL	155
9.25.2.21	RS_C1	156
9.25.2.22	SHORT	156
9.25.3	Typedef Documentation	156
9.25.3.1	dCOOmat	156
9.25.3.2	dCSRLmat	156
9.25.3.3	dCSRmat	156
9.25.3.4	ddenmat	156
9.25.3.5	dSTRmat	156
9.25.3.6	dvector	156
9.25.3.7	grid2d	156
9.25.3.8	iCOOmat	156
9.25.3.9	iCSRmat	157
9.25.3.10	idenmat	157
9.25.3.11	ivector	157
9.25.3.12	LinkList	157
9.25.3.13	ListElement	157
9.25.3.14	pcgrid2d	157
9.25.3.15	pgrid2d	157
9.25.4	Variable Documentation	157
9.25.4.1	count	157
9.25.4.2	IMAP	157
9.25.4.3	MAXIMAP	157

9.25.4.4	<code>nx_rb</code>	158
9.25.4.5	<code>ny_rb</code>	158
9.25.4.6	<code>nz_rb</code>	158
9.25.4.7	<code>total_alloc_count</code>	158
9.25.4.8	<code>total_alloc_mem</code>	158
9.26	<code>fasp_block.h</code> File Reference	158
9.26.1	Detailed Description	159
9.26.2	Typedef Documentation	159
9.26.2.1	<code>block_BSR</code>	159
9.26.2.2	<code>block_dCSRmat</code>	159
9.26.2.3	<code>block_dvector</code>	160
9.26.2.4	<code>block_iCSRmat</code>	160
9.26.2.5	<code>block_ivector</code>	160
9.26.2.6	<code>block_Reservoir</code>	160
9.26.2.7	<code>dBSRmat</code>	160
9.26.2.8	<code>precond_block_reservoir_data</code>	160
9.27	<code>fasp_const.h</code> File Reference	160
9.27.1	Detailed Description	163
9.27.2	Macro Definition Documentation	164
9.27.2.1	<code>AMLI_CYCLE</code>	164
9.27.2.2	<code>ASCEND</code>	164
9.27.2.3	<code>BIGREAL</code>	164
9.27.2.4	<code>CF_ORDER</code>	164
9.27.2.5	<code>CGPT</code>	164
9.27.2.6	<code>CLASSIC_AMG</code>	164
9.27.2.7	<code>COARSE_AC</code>	165
9.27.2.8	<code>COARSE_CR</code>	165
9.27.2.9	<code>COARSE_MIS</code>	165
9.27.2.10	<code>COARSE_RS</code>	165
9.27.2.11	<code>COARSE_RSP</code>	165
9.27.2.12	<code>CPFIRST</code>	165
9.27.2.13	<code>DESCEND</code>	165
9.27.2.14	<code>ERROR_ALLOC_MEM</code>	165
9.27.2.15	<code>ERROR_AMG_COARSE_TYPE</code>	166
9.27.2.16	<code>ERROR_AMG_COARSEING</code>	166
9.27.2.17	<code>ERROR_AMG_INTERP_TYPE</code>	166
9.27.2.18	<code>ERROR_AMG_SMOOTH_TYPE</code>	166

9.27.2.19 ERROR_DATA_STRUCTURE	166
9.27.2.20 ERROR_DATA_ZERODIAG	166
9.27.2.21 ERROR_DUMMY_VAR	166
9.27.2.22 ERROR_INPUT_PAR	166
9.27.2.23 ERROR_LIC_TYPE	166
9.27.2.24 ERROR_MAT_SIZE	167
9.27.2.25 ERROR_MISC	167
9.27.2.26 ERROR_NUM_BLOCKS	167
9.27.2.27 ERROR_OPEN_FILE	167
9.27.2.28 ERROR_QUAD_DIM	167
9.27.2.29 ERROR_QUAD_TYPE	167
9.27.2.30 ERROR_REGRESS	167
9.27.2.31 ERROR_SOLVER_EXIT	167
9.27.2.32 ERROR_SOLVER_ILUSETUP	167
9.27.2.33 ERROR_SOLVER_MAXIT	168
9.27.2.34 ERROR_SOLVER_MISC	168
9.27.2.35 ERROR_SOLVER_PRECTYPE	168
9.27.2.36 ERROR_SOLVER_SOLSTAG	168
9.27.2.37 ERROR_SOLVER_STAG	168
9.27.2.38 ERROR_SOLVER_TOLSMALL	168
9.27.2.39 ERROR_SOLVER_TYPE	168
9.27.2.40 ERROR_UNKNOWN	168
9.27.2.41 ERROR_WRONG_FILE	168
9.27.2.42 FALSE	169
9.27.2.43 FASP_SUCCESS	169
9.27.2.44 FGPT	169
9.27.2.45 FPFIRST	169
9.27.2.46 G0PT	169
9.27.2.47 ILUk	169
9.27.2.48 ILUt	169
9.27.2.49 ILUtp	169
9.27.2.50 INTERP_DIR	170
9.27.2.51 INTERP_ENG	170
9.27.2.52 INTERP_STD	170
9.27.2.53 ISPT	170
9.27.2.54 MAT_bBSR	170
9.27.2.55 MAT_bCSR	170

9.27.2.56 MAT_BSR	170
9.27.2.57 MAT_CSR	170
9.27.2.58 MAT_CSRL	171
9.27.2.59 MAT_FREE	171
9.27.2.60 MAT_STR	171
9.27.2.61 MAT_SymCSR	171
9.27.2.62 MAX_AMG_LVL	171
9.27.2.63 MAX_CRATE	171
9.27.2.64 MAX_REFINE_LVL	171
9.27.2.65 MAX_RESTART	171
9.27.2.66 MAX_STAG	172
9.27.2.67 MIN_CDOF	172
9.27.2.68 MIN_CRATE	172
9.27.2.69 NL_AMLI_CYCLE	172
9.27.2.70 NO_ORDER	172
9.27.2.71 OFF	172
9.27.2.72 ON	172
9.27.2.73 OPENMP_HOLDS	172
9.27.2.74 PAIRWISE	173
9.27.2.75 PREC_AMG	173
9.27.2.76 PREC_DIAG	173
9.27.2.77 PREC_FMG	173
9.27.2.78 PREC_ILU	173
9.27.2.79 PREC_NULL	173
9.27.2.80 PREC_SCHWARZ	173
9.27.2.81 PRINT_ALL	173
9.27.2.82 PRINT_MIN	174
9.27.2.83 PRINT_MORE	174
9.27.2.84 PRINT_MOST	174
9.27.2.85 PRINT_NONE	174
9.27.2.86 PRINT_SOME	174
9.27.2.87 SA_AMG	174
9.27.2.88 SCHWARZ_BACKWARD	174
9.27.2.89 SCHWARZ_FORWARD	174
9.27.2.90 SCHWARZ_SYMMETRIC	175
9.27.2.91 SMALLREAL	175
9.27.2.92 SMALLREAL2	175

9.27.2.93 SMOOTHER_BLKOil	175
9.27.2.94 SMOOTHER_CG	175
9.27.2.95 SMOOTHER_GS	175
9.27.2.96 SMOOTHER_GSOR	175
9.27.2.97 SMOOTHER_JACOBI	175
9.27.2.98 SMOOTHER_L1DIAG	176
9.27.2.99 SMOOTHER_POLY	176
9.27.2.100 SMOOTHER_SGS	176
9.27.2.101 SMOOTHER_SGSOR	176
9.27.2.102 SMOOTHER_SOR	176
9.27.2.103 SMOOTHER_SPETEN	176
9.27.2.104 SMOOTHER_SSOR	176
9.27.2.105 SOLVER_AMG	176
9.27.2.106 SOLVER_BiCGstab	176
9.27.2.107 SOLVER_CG	177
9.27.2.108 SOLVER_DEFAULT	177
9.27.2.109 SOLVER_FMG	177
9.27.2.110 SOLVER_GCG	177
9.27.2.111 SOLVER_GCR	177
9.27.2.112 SOLVER_GMRES	177
9.27.2.113 SOLVER_MinRes	177
9.27.2.114 SOLVER_MUMPS	177
9.27.2.115 SOLVER_SBiCGstab	178
9.27.2.116 SOLVER_SCG	178
9.27.2.117 SOLVER_SGCG	178
9.27.2.118 SOLVER_SGMRES	178
9.27.2.119 SOLVER_SMinRes	178
9.27.2.120 SOLVER_SUPERLU	178
9.27.2.121 SOLVER_SVFGMRES	178
9.27.2.122 SOLVER_SVGMRES	178
9.27.2.123 SOLVER_UMFPACK	178
9.27.2.124 SOLVER_VFGMRES	179
9.27.2.125 SOLVER_VGMRES	179
9.27.2.126 STAG_RATIO	179
9.27.2.127 STOP_MOD_REL_RES	179
9.27.2.128 STOP_REL_PRECRES	179
9.27.2.129 STOP_REL_RES	179

9.27.2.130	TRUE	179
9.27.2.131	UA_AMG	179
9.27.2.132	UNPT	180
9.27.2.133	USERDEFINED	180
9.27.2.134	V_CYCLE	180
9.27.2.135	VMB	180
9.27.2.136	W_CYCLE	180
9.28	fmgcycle.c File Reference	180
9.28.1	Detailed Description	181
9.28.2	Function Documentation	181
9.28.2.1	fasp_solver_fmgcycle(AMG_data *mgl, AMG_param *param)	181
9.29	formats.c File Reference	181
9.29.1	Detailed Description	182
9.29.2	Function Documentation	182
9.29.2.1	fasp_format_bdcsr_dcsr(block_dCSRmat *Ab)	182
9.29.2.2	fasp_format_dbsr_dcoo(dBSRmat *B)	182
9.29.2.3	fasp_format_dbsr_dcsr(dBSRmat *B)	183
9.29.2.4	fasp_format_dcoo_dcsr(dCOOmat *A, dCSRmat *B)	183
9.29.2.5	fasp_format_dcsr_dbsr(dCSRmat *A, const INT nb)	184
9.29.2.6	fasp_format_dcsr_dcoo(dCSRmat *A, dCOOmat *B)	185
9.29.2.7	fasp_format_dcsr_dcsr(dCSRmat *A)	185
9.29.2.8	fasp_format_dstr_dbsr(dSTRmat *B)	186
9.29.2.9	fasp_format_dstr_dcsr(dSTRmat *A, dCSRmat *B)	186
9.30	givens.c File Reference	187
9.30.1	Detailed Description	187
9.30.2	Function Documentation	187
9.30.2.1	fasp_aux_givens(const REAL beta, dCSRmat *H, dvector *y, REAL *tmp)	187
9.31	gmh_poisson.c File Reference	188
9.31.1	Detailed Description	188
9.31.2	Function Documentation	188
9.31.2.1	fasp_poisson_fmgh_1D(REAL *u, REAL *b, INT nx, INT maxlevel, REAL rtol, const SHORT prtlvl)	188
9.31.2.2	fasp_poisson_fmgh_2D(REAL *u, REAL *b, INT nx, INT ny, INT maxlevel, REAL rtol, const SHORT prtlvl)	189
9.31.2.3	fasp_poisson_fmgh_3D(REAL *u, REAL *b, INT nx, INT ny, INT nz, INT maxlevel, REAL rtol, const SHORT prtlvl)	189
9.31.2.4	fasp_poisson_gmh_1D(REAL *u, REAL *b, const INT nx, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	190

9.31.2.5	<code>fasp_poisson_gmg_2D</code> (REAL *u, REAL *b, const INT nx, const INT ny, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	190
9.31.2.6	<code>fasp_poisson_gmg_3D</code> (REAL *u, REAL *b, const INT nx, const INT ny, const INT nz, const INT maxlevel, const REAL rtol, const SHORT prtlvl)	191
9.31.2.7	<code>fasp_poisson_pcg_gmg_1D</code> (REAL *u, REAL *b, INT nx, INT maxlevel, REAL rtol, const SHORT prtlvl)	192
9.31.2.8	<code>fasp_poisson_pcg_gmg_2D</code> (REAL *u, REAL *b, INT nx, INT ny, INT maxlevel, REAL rtol, const SHORT prtlvl)	193
9.31.2.9	<code>fasp_poisson_pcg_gmg_3D</code> (REAL *u, REAL *b, INT nx, INT ny, INT nz, INT maxlevel, REAL rtol, const SHORT prtlvl)	193
9.32	graphics.c File Reference	194
9.32.1	Detailed Description	194
9.32.2	Function Documentation	195
9.32.2.1	<code>fasp_dbsr_plot</code> (const dBSRmat *A, const char *fname)	195
9.32.2.2	<code>fasp_dbsr_subplot</code> (const dBSRmat *A, const char *filename, INT size)	195
9.32.2.3	<code>fasp_dcsr_plot</code> (const dCSRmat *A, const char *fname)	196
9.32.2.4	<code>fasp_dcsr_subplot</code> (const dCSRmat *A, const char *filename, INT size)	196
9.32.2.5	<code>fasp_grid2d_plot</code> (pgrid2d pg, INT level)	197
9.33	ilu.f File Reference	198
9.33.1	Detailed Description	198
9.34	ilu_setup_bsr.c File Reference	198
9.34.1	Detailed Description	199
9.34.2	Function Documentation	199
9.34.2.1	<code>fasp_ilu_dbsr_setup</code> (dBSRmat *A, ILU_data *iludata, ILU_param *iluparam)	199
9.35	ilu_setup_csr.c File Reference	199
9.35.1	Detailed Description	200
9.35.2	Function Documentation	200
9.35.2.1	<code>fasp_ilu_dcsr_setup</code> (dCSRmat *A, ILU_data *iludata, ILU_param *iluparam)	200
9.36	ilu_setup_str.c File Reference	200
9.36.1	Detailed Description	201
9.36.2	Function Documentation	201
9.36.2.1	<code>fasp_ilu_dstr_setup0</code> (dSTRmat *A, dSTRmat *LU)	201
9.36.2.2	<code>fasp_ilu_dstr_setup1</code> (dSTRmat *A, dSTRmat *LU)	201
9.37	init.c File Reference	202
9.37.1	Detailed Description	202
9.37.2	Function Documentation	203
9.37.2.1	<code>fasp_amg_data_bsr_create</code> (SHORT max_levels)	203
9.37.2.2	<code>fasp_amg_data_bsr_free</code> (AMG_data_bsr *mgf)	204

9.37.2.3	<code>fasp_amg_data_create(SHORT max_levels)</code>	204
9.37.2.4	<code>fasp_amg_data_free(AMG_data *mgl, AMG_param *param)</code>	205
9.37.2.5	<code>fasp_ilu_data_alloc(INT iwk, INT nwork, ILU_data *iludata)</code>	206
9.37.2.6	<code>fasp_ilu_data_free(ILU_data *ILUdata)</code>	206
9.37.2.7	<code>fasp_ilu_data_null(ILU_data *ILUdata)</code>	207
9.37.2.8	<code>fasp_precond_data_null(precond_data *pcdata)</code>	208
9.37.2.9	<code>fasp_precond_null(precond *pcdata)</code>	208
9.37.2.10	<code>fasp_Schwarz_data_free(Schwarz_data *Schwarz)</code>	208
9.38	input.c File Reference	209
9.38.1	Detailed Description	209
9.38.2	Function Documentation	209
9.38.2.1	<code>fasp_param_check(input_param *inparam)</code>	209
9.38.2.2	<code>fasp_param_input(const char *filenm, input_param *inparam)</code>	210
9.39	interface_mumps.c File Reference	210
9.39.1	Detailed Description	210
9.39.2	Function Documentation	211
9.39.2.1	<code>fasp_solver_mumps(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)</code>	211
9.39.2.2	<code>fasp_solver_mumps_steps(dCSRmat *ptrA, dvector *b, dvector *u, Mumps_data *mumps)</code>	212
9.40	interface_samg.c File Reference	212
9.40.1	Detailed Description	213
9.40.2	Function Documentation	213
9.40.2.1	<code>dCSRmat2SAMGInput(dCSRmat *A, char *filefrm, char *fileamg)</code>	213
9.40.2.2	<code>dvector2SAMGInput(dvector *vec, char *filename)</code>	213
9.41	interface_superlu.c File Reference	214
9.41.1	Detailed Description	214
9.41.2	Function Documentation	214
9.41.2.1	<code>fasp_solver_superlu(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)</code>	214
9.42	interface_umfpack.c File Reference	214
9.42.1	Detailed Description	215
9.42.2	Function Documentation	215
9.42.2.1	<code>fasp_solver_umfpack(dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)</code>	215
9.43	interpolation.c File Reference	215
9.43.1	Detailed Description	216
9.43.2	Function Documentation	216
9.43.2.1	<code>fasp_amg_interp(dCSRmat *A, ivector *vertices, dCSRmat *P, iCSRmat *S, AMG_param *param)</code>	216

9.43.2.2	fasp_amg_interp1(dCSRmat *A, ivecator *vertices, dCSRmat *P, AMG_param *param, iCSRmat *S, INT *icor_ysk)	216
9.43.2.3	fasp_amg_interp_trunc(dCSRmat *P, AMG_param *param)	217
9.44	interpolation_em.c File Reference	217
9.44.1	Detailed Description	218
9.44.2	Function Documentation	218
9.44.2.1	fasp_amg_interp_em(dCSRmat *A, ivecator *vertices, dCSRmat *P, AMG_param *param)	218
9.45	io.c File Reference	218
9.45.1	Detailed Description	220
9.45.2	Function Documentation	220
9.45.2.1	fasp_dbsr_print(dBSRmat *A)	220
9.45.2.2	fasp_dbsr_read(const char *filename, dBSRmat *A)	221
9.45.2.3	fasp_dbsr_write(const char *filename, dBSRmat *A)	221
9.45.2.4	fasp_dbsr_write_coo(const char *filename, const dBSRmat *A)	222
9.45.2.5	fasp_dcoo1_read(const char *filename, dCOOmat *A)	222
9.45.2.6	fasp_dcoo_print(dCOOmat *A)	223
9.45.2.7	fasp_dcoo_read(const char *filename, dCSRmat *A)	223
9.45.2.8	fasp_dcoo_shift_read(const char *filename, dCSRmat *A)	224
9.45.2.9	fasp_dcoo_write(const char *filename, dCSRmat *A)	224
9.45.2.10	fasp_dcsr_print(dCSRmat *A)	225
9.45.2.11	fasp_dcsr_read(const char *filename, dCSRmat *A)	225
9.45.2.12	fasp_dcsr_write_coo(const char *filename, const dCSRmat *A)	225
9.45.2.13	fasp_dcsrvec1_read(const char *filename, dCSRmat *A, dvector *b)	226
9.45.2.14	fasp_dcsrvec1_write(const char *filename, dCSRmat *A, dvector *b)	226
9.45.2.15	fasp_dcsrvec2_read(const char *filemat, const char *filerhs, dCSRmat *A, dvector *b)	227
9.45.2.16	fasp_dcsrvec2_write(const char *filemat, const char *filerhs, dCSRmat *A, dvector *b)	228
9.45.2.17	fasp_dmtx_read(const char *filename, dCSRmat *A)	228
9.45.2.18	fasp_dmtxsym_read(const char *filename, dCSRmat *A)	229
9.45.2.19	fasp_dstr_print(dSTRmat *A)	229
9.45.2.20	fasp_dstr_read(const char *filename, dSTRmat *A)	230
9.45.2.21	fasp_dstr_write(const char *filename, dSTRmat *A)	230
9.45.2.22	fasp_dvec_print(INT n, dvector *u)	231
9.45.2.23	fasp_dvec_read(const char *filename, dvector *b)	231
9.45.2.24	fasp_dvec_write(const char *filename, dvector *vec)	232
9.45.2.25	fasp_dvecind_read(const char *filename, dvector *b)	232
9.45.2.26	fasp_dvecind_write(const char *filename, dvector *vec)	233

9.45.2.27	<code>fasp_hb_read(const char *input_file, dCSRmat *A, dvector *b)</code>	234
9.45.2.28	<code>fasp_ivec_print(INT n, ivec *u)</code>	234
9.45.2.29	<code>fasp_ivec_read(const char *filename, ivec *b)</code>	235
9.45.2.30	<code>fasp_ivec_write(const char *filename, ivec *vec)</code>	235
9.45.2.31	<code>fasp_ivecind_read(const char *filename, ivec *b)</code>	236
9.45.2.32	<code>fasp_matrix_read(const char *filename, void *A)</code>	236
9.45.2.33	<code>fasp_matrix_read_bin(const char *filename, void *A)</code>	237
9.45.2.34	<code>fasp_matrix_write(const char *filename, void *A, INT flag)</code>	238
9.45.2.35	<code>fasp_vector_read(const char *filerhs, void *b)</code>	239
9.45.2.36	<code>fasp_vector_write(const char *filerhs, void *b, INT flag)</code>	240
9.45.3	Variable Documentation	241
9.45.3.1	<code>dlength</code>	241
9.45.3.2	<code>ilength</code>	241
9.46	<code>itsolver_bcsr.c</code> File Reference	241
9.46.1	Detailed Description	241
9.46.2	Function Documentation	242
9.46.2.1	<code>fasp_solver_bdcsr_itsolver(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	242
9.46.2.2	<code>fasp_solver_bdcsr_krylov(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	243
9.46.2.3	<code>fasp_solver_bdcsr_krylov_block_3(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)</code>	243
9.46.2.4	<code>fasp_solver_bdcsr_krylov_block_4(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, dCSRmat *A_diag)</code>	244
9.46.2.5	<code>fasp_solver_bdcsr_krylov_sweeping(block_dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, INT NumLayers, block_dCSRmat *Ai, dCSRmat *local_A, ivec *local_index)</code>	245
9.47	<code>itsolver_bsr.c</code> File Reference	245
9.47.1	Detailed Description	246
9.47.2	Function Documentation	246
9.47.2.1	<code>fasp_solver_dbsr_itsolver(dBSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	246
9.47.2.2	<code>fasp_solver_dbsr_krylov(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	246
9.47.2.3	<code>fasp_solver_dbsr_krylov_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam)</code>	247
9.47.2.4	<code>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)</code>	247
9.47.2.5	<code>fasp_solver_dbsr_krylov_diag(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	248

9.47.2.6	<code>fasp_solver_dbsr_krylov_ilu(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)</code>	248
9.47.2.7	<code>fasp_solver_dbsr_krylov_nk_amg(dBSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam, const INT nk_dim, dvector *nk)</code>	249
9.48	<code>itsolver_csr.c</code> File Reference	250
9.48.1	Detailed Description	250
9.48.2	Function Documentation	250
9.48.2.1	<code>fasp_solver_dcsr_itsolver(dCSRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	250
9.48.2.2	<code>fasp_solver_dcsr_krylov(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	251
9.48.2.3	<code>fasp_solver_dcsr_krylov_amg(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, AMG_param *amgparam)</code>	251
9.48.2.4	<code>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)</code>	252
9.48.2.5	<code>fasp_solver_dcsr_krylov_diag(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	252
9.48.2.6	<code>fasp_solver_dcsr_krylov_ilu(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)</code>	253
9.48.2.7	<code>fasp_solver_dcsr_krylov_ilu_M(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam, dCSRmat *M)</code>	253
9.48.2.8	<code>fasp_solver_dcsr_krylov_Schwarz(dCSRmat *A, dvector *b, dvector *x, itsolver_param *itparam, Schwarz_param *schparam)</code>	254
9.49	<code>itsolver_mf.c</code> File Reference	255
9.49.1	Detailed Description	255
9.49.2	Function Documentation	255
9.49.2.1	<code>fasp_solver_itsolver(mxv_matfree *mf, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	255
9.49.2.2	<code>fasp_solver_itsolver_init(INT matrix_format, mxv_matfree *mf, void *A)</code>	256
9.49.2.3	<code>fasp_solver_krylov(mxv_matfree *mf, dvector *b, dvector *x, itsolver_param *itparam)</code>	256
9.50	<code>itsolver_str.c</code> File Reference	257
9.50.1	Detailed Description	257
9.50.2	Function Documentation	257
9.50.2.1	<code>fasp_solver_dstr_itsolver(dSTRmat *A, dvector *b, dvector *x, precondition *pc, itsolver_param *itparam)</code>	257
9.50.2.2	<code>fasp_solver_dstr_krylov(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	258
9.50.2.3	<code>fasp_solver_dstr_krylov_blockgs(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ivector *neigh, ivector *order)</code>	258
9.50.2.4	<code>fasp_solver_dstr_krylov_diag(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam)</code>	259
9.50.2.5	<code>fasp_solver_dstr_krylov_ilu(dSTRmat *A, dvector *b, dvector *x, itsolver_param *itparam, ILU_param *iluparam)</code>	259

9.51	lu.c File Reference	260
9.51.1	Detailed Description	260
9.51.2	Function Documentation	260
9.51.2.1	fasp_smat_lu_decomp(<code>REAL *A</code> , <code>INT pivot[]</code> , <code>INT n</code>)	260
9.51.2.2	fasp_smat_lu_solve(<code>REAL *A</code> , <code>REAL b[]</code> , <code>INT pivot[]</code> , <code>REAL x[]</code> , <code>INT n</code>)	261
9.52	memory.c File Reference	262
9.52.1	Detailed Description	262
9.52.2	Function Documentation	262
9.52.2.1	fasp_mem_calloc(<code>LONGLONG size</code> , <code>INT type</code>)	262
9.52.2.2	fasp_mem_check(<code>void *ptr</code> , <code>const char *message</code> , <code>INT ERR</code>)	263
9.52.2.3	fasp_mem_dcsr_check(<code>dCSRmat *A</code>)	263
9.52.2.4	fasp_mem_free(<code>void *mem</code>)	264
9.52.2.5	fasp_mem_iludata_check(<code>ILU_data *iludata</code>)	264
9.52.2.6	fasp_mem_realloc(<code>void *oldmem</code> , <code>LONGLONG tsize</code>)	265
9.52.2.7	fasp_mem_usage()	265
9.52.3	Variable Documentation	265
9.52.3.1	total_alloc_count	265
9.52.3.2	total_alloc_mem	266
9.53	message.c File Reference	266
9.53.1	Detailed Description	266
9.53.2	Function Documentation	266
9.53.2.1	fasp_chkerr(<code>const SHORT status</code> , <code>const char *fctname</code>)	266
9.53.2.2	print_amgcomplexity(<code>AMG_data *mgl</code> , <code>const SHORT prtlvl</code>)	267
9.53.2.3	print_amgcomplexity_bsr(<code>AMG_data_bsr *mgl</code> , <code>const SHORT prtlvl</code>)	267
9.53.2.4	print_cputime(<code>const char *message</code> , <code>const REAL cputime</code>)	267
9.53.2.5	print_itinfo(<code>const INT prtlvl</code> , <code>const INT stop_type</code> , <code>const INT iter</code> , <code>const REAL relres</code> , <code>const REAL absres</code> , <code>const REAL factor</code>)	268
9.53.2.6	print_message(<code>const INT prtlvl</code> , <code>const char *message</code>)	268
9.54	mgcycle.c File Reference	269
9.54.1	Detailed Description	269
9.54.2	Function Documentation	269
9.54.2.1	fasp_solver_mgcycle(<code>AMG_data *mgl</code> , <code>AMG_param *param</code>)	269
9.54.2.2	fasp_solver_mgcycle_bsr(<code>AMG_data_bsr *mgl</code> , <code>AMG_param *param</code>)	270
9.55	mgrecur.c File Reference	271
9.55.1	Detailed Description	271
9.55.2	Function Documentation	271
9.55.2.1	fasp_solver_mgrecur(<code>AMG_data *mgl</code> , <code>AMG_param *param</code> , <code>INT level</code>)	271

9.56 ordering.c File Reference	272
9.56.1 Detailed Description	272
9.56.2 Function Documentation	273
9.56.2.1 fasp_aux_dQuickSort(REAL *a, INT left, INT right)	273
9.56.2.2 fasp_aux_dQuickSortIndex(REAL *a, INT left, INT right, INT *index)	273
9.56.2.3 fasp_aux_iQuickSort(INT *a, INT left, INT right)	273
9.56.2.4 fasp_aux_iQuickSortIndex(INT *a, INT left, INT right, INT *index)	274
9.56.2.5 fasp_aux_merge(INT numbers[], INT work[], INT left, INT mid, INT right)	274
9.56.2.6 fasp_aux_msort(INT numbers[], INT work[], INT left, INT right)	275
9.56.2.7 fasp_aux_unique(INT numbers[], INT size)	275
9.56.2.8 fasp_BinarySearch(INT *list, INT value, INT nlist)	276
9.56.2.9 fasp_dcsr_CMK_order(const dCSRmat *A, INT *order, INT *oindex)	276
9.56.2.10 fasp_dcsr_RCMK_order(const dCSRmat *A, INT *order, INT *oindex, INT *rorder)	277
9.57 parameters.c File Reference	277
9.57.1 Detailed Description	278
9.57.2 Function Documentation	278
9.57.2.1 fasp_param_amg_init(AMG_param *amgparam)	278
9.57.2.2 fasp_param_amg_print(AMG_param *param)	279
9.57.2.3 fasp_param_amg_set(AMG_param *param, input_param *iniparam)	279
9.57.2.4 fasp_param_amg_to_prec(precond_data *pcdata, AMG_param *amgparam)	279
9.57.2.5 fasp_param_amg_to_prec_bsr(precond_data_bsr *pcdata, AMG_param *amgparam)	280
9.57.2.6 fasp_param_ilu_init(ILU_param *iluparam)	280
9.57.2.7 fasp_param_ilu_print(ILU_param *param)	280
9.57.2.8 fasp_param_ilu_set(ILU_param *iluparam, input_param *iniparam)	281
9.57.2.9 fasp_param_init(input_param *iniparam, itsolver_param *itsparam, AMG_param *amgparam, ILU_param *iluparam, Schwarz_param *schparam)	281
9.57.2.10 fasp_param_input_init(input_param *iniparam)	282
9.57.2.11 fasp_param_prec_to_amg(AMG_param *amgparam, precondition_data *pcdata)	282
9.57.2.12 fasp_param_prec_to_amg_bsr(AMG_param *amgparam, precondition_data_bsr *pcdata)	282
9.57.2.13 fasp_param_Schwarz_init(Schwarz_param *schparam)	283
9.57.2.14 fasp_param_Schwarz_print(Schwarz_param *param)	284
9.57.2.15 fasp_param_Schwarz_set(Schwarz_param *schparam, input_param *iniparam)	284
9.57.2.16 fasp_param_set(int argc, const char *argv[], input_param *iniparam)	284
9.57.2.17 fasp_param_solver_init(itsolver_param *itsparam)	285
9.57.2.18 fasp_param_solver_print(itsolver_param *param)	285
9.57.2.19 fasp_param_solver_set(itsolver_param *itsparam, input_param *iniparam)	285
9.58 pbcgs.c File Reference	286

9.58.1 Detailed Description	286
9.58.2 Function Documentation	287
9.58.2.1 fasp_solver_bdcsl_pbcgs(block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	287
9.58.2.2 fasp_solver_dbsl_pbcgs(dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	288
9.58.2.3 fasp_solver_dcsr_pbcgs(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	289
9.58.2.4 fasp_solver_dstr_pbcgs(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	289
9.59 pbcgs_mf.c File Reference	290
9.59.1 Detailed Description	290
9.59.2 Function Documentation	291
9.59.2.1 fasp_solver_pbcgs(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	291
9.60 pcg.c File Reference	292
9.60.1 Detailed Description	293
9.60.2 Function Documentation	294
9.60.2.1 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)	294
9.60.2.2 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)	295
9.60.2.3 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)	296
9.60.2.4 fasp_solver_dstr_pcg(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	297
9.61 pcg_mf.c File Reference	298
9.61.1 Detailed Description	298
9.61.2 Function Documentation	299
9.61.2.1 fasp_solver_pcg(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	299
9.62 pgcg.c File Reference	300
9.62.1 Detailed Description	300
9.62.2 Function Documentation	300
9.62.2.1 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)	300
9.63 pgcg_mf.c File Reference	301
9.63.1 Detailed Description	301
9.63.2 Function Documentation	301
9.63.2.1 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)	301

9.64	pgcr.c File Reference	302
9.64.1	Detailed Description	302
9.64.2	Function Documentation	303
9.64.2.1	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)	303
9.64.2.2	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)	304
9.65	pgmres.c File Reference	305
9.65.1	Detailed Description	305
9.65.2	Function Documentation	305
9.65.2.1	fasp_solver_bdcsr_pgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	305
9.65.2.2	fasp_solver_dbsr_pgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	306
9.65.2.3	fasp_solver_dcsr_pgmres(dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	307
9.65.2.4	fasp_solver_dstr_pgmres(dSTRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	307
9.66	pgmres_mf.c File Reference	308
9.66.1	Detailed Description	308
9.66.2	Function Documentation	309
9.66.2.1	fasp_solver_pgmres(mxv_matfree *mf, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtvl)	309
9.67	pminres.c File Reference	309
9.67.1	Detailed Description	310
9.67.2	Function Documentation	311
9.67.2.1	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)	311
9.67.2.2	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)	311
9.67.2.3	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)	312
9.68	pminres_mf.c File Reference	313
9.68.1	Detailed Description	313
9.68.2	Function Documentation	314

9.68.2.1	fasp_solver_pminres(mxv_matfree *mf, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl)	314
9.69	precond_bcsr.c File Reference	315
9.69.1	Detailed Description	315
9.69.2	Function Documentation	316
9.69.2.1	fasp_precond_block_diag_3(double *r, double *z, void *data)	316
9.69.2.2	fasp_precond_block_diag_3_amg(double *r, double *z, void *data)	317
9.69.2.3	fasp_precond_block_diag_4(double *r, double *z, void *data)	317
9.69.2.4	fasp_precond_block_lower_3(double *r, double *z, void *data)	318
9.69.2.5	fasp_precond_block_lower_3_amg(double *r, double *z, void *data)	319
9.69.2.6	fasp_precond_block_lower_4(double *r, double *z, void *data)	319
9.69.2.7	fasp_precond_block_SGS_3(double *r, double *z, void *data)	320
9.69.2.8	fasp_precond_block_SGS_3_amg(double *r, double *z, void *data)	321
9.69.2.9	fasp_precond_block_upper_3(double *r, double *z, void *data)	321
9.69.2.10	fasp_precond_block_upper_3_amg(double *r, double *z, void *data)	322
9.69.2.11	fasp_precond_sweeping(double *r, double *z, void *data)	323
9.70	precond_bsr.c File Reference	323
9.70.1	Detailed Description	324
9.70.2	Function Documentation	324
9.70.2.1	fasp_precond_dbsr_amg(REAL *r, REAL *z, void *data)	324
9.70.2.2	fasp_precond_dbsr_amg_nk(REAL *r, REAL *z, void *data)	324
9.70.2.3	fasp_precond_dbsr_diag(REAL *r, REAL *z, void *data)	325
9.70.2.4	fasp_precond_dbsr_diag_nc2(REAL *r, REAL *z, void *data)	325
9.70.2.5	fasp_precond_dbsr_diag_nc3(REAL *r, REAL *z, void *data)	326
9.70.2.6	fasp_precond_dbsr_diag_nc5(REAL *r, REAL *z, void *data)	327
9.70.2.7	fasp_precond_dbsr_diag_nc7(REAL *r, REAL *z, void *data)	328
9.70.2.8	fasp_precond_dbsr_ilu(REAL *r, REAL *z, void *data)	329
9.70.2.9	fasp_precond_dbsr_nli(REAL *r, REAL *z, void *data)	330
9.71	precond_csr.c File Reference	330
9.71.1	Detailed Description	331
9.71.2	Function Documentation	331
9.71.2.1	fasp_precond_amg(REAL *r, REAL *z, void *data)	331
9.71.2.2	fasp_precond_amg_nk(REAL *r, REAL *z, void *data)	332
9.71.2.3	fasp_precond_nli(REAL *r, REAL *z, void *data)	332
9.71.2.4	fasp_precond_diag(REAL *r, REAL *z, void *data)	332
9.71.2.5	fasp_precond_famg(REAL *r, REAL *z, void *data)	333
9.71.2.6	fasp_precond_free(SHORT precondition_type, precondition *pc)	333

9.71.2.7	<code>fasp_precond_ilu(REAL *r, REAL *z, void *data)</code>	334
9.71.2.8	<code>fasp_precond_ilu_backward(REAL *r, REAL *z, void *data)</code>	334
9.71.2.9	<code>fasp_precond_ilu_forward(REAL *r, REAL *z, void *data)</code>	334
9.71.2.10	<code>fasp_precond_nl_amli(REAL *r, REAL *z, void *data)</code>	335
9.71.2.11	<code>fasp_precond_Schwarz(REAL *r, REAL *z, void *data)</code>	335
9.71.2.12	<code>fasp_precond_setup(SHORT precondition_type, AMG_param *amgparam, ILU_param *iluparam, dCSRmat *A)</code>	335
9.72	<code>precond_str.c</code> File Reference	336
9.72.1	Detailed Description	337
9.72.2	Function Documentation	337
9.72.2.1	<code>fasp_precond_dstr_blockgs(REAL *r, REAL *z, void *data)</code>	337
9.72.2.2	<code>fasp_precond_dstr_diag(REAL *r, REAL *z, void *data)</code>	337
9.72.2.3	<code>fasp_precond_dstr_ilu0(REAL *r, REAL *z, void *data)</code>	337
9.72.2.4	<code>fasp_precond_dstr_ilu0_backward(REAL *r, REAL *z, void *data)</code>	338
9.72.2.5	<code>fasp_precond_dstr_ilu0_forward(REAL *r, REAL *z, void *data)</code>	338
9.72.2.6	<code>fasp_precond_dstr_ilu1(REAL *r, REAL *z, void *data)</code>	339
9.72.2.7	<code>fasp_precond_dstr_ilu1_backward(REAL *r, REAL *z, void *data)</code>	340
9.72.2.8	<code>fasp_precond_dstr_ilu1_forward(REAL *r, REAL *z, void *data)</code>	340
9.73	<code>pvfgmres.c</code> File Reference	341
9.73.1	Detailed Description	341
9.73.2	Function Documentation	341
9.73.2.1	<code>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 prtlvl)</code>	341
9.73.2.2	<code>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)</code>	342
9.73.2.3	<code>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)</code>	343
9.74	<code>pvfgmres_mf.c</code> File Reference	343
9.74.1	Detailed Description	344
9.74.2	Function Documentation	344
9.74.2.1	<code>fasp_solver_pvfgmres(mvx_matfree *mf, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)</code>	344
9.75	<code>pvgmres.c</code> File Reference	345
9.75.1	Detailed Description	345
9.75.2	Function Documentation	345

9.75.2.1	<code>fasp_solver_bdcslr_pvgmres(block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)</code>	345
9.75.2.2	<code>fasp_solver_dbsl_r_pvgmres(dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)</code>	346
9.75.2.3	<code>fasp_solver_dcsl_r_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)</code>	347
9.75.2.4	<code>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)</code>	348
9.76	<code>pvgmres_mf.c</code> File Reference	349
9.76.1	Detailed Description	349
9.76.2	Function Documentation	349
9.76.2.1	<code>fasp_solver_pvgmres(mxv_matfree *mf, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, SHORT restart, const SHORT stop_type, const SHORT prtlvl)</code>	349
9.77	<code>quadrature.c</code> File Reference	350
9.77.1	Detailed Description	350
9.77.2	Function Documentation	350
9.77.2.1	<code>fasp_gauss2d(INT num_qp, INT ncoor, REAL(*gauss)[3])</code>	350
9.77.2.2	<code>fasp_quad2d(INT num_qp, INT ncoor, REAL(*quad)[3])</code>	351
9.78	<code>rap.c</code> File Reference	351
9.78.1	Detailed Description	352
9.78.2	Function Documentation	352
9.78.2.1	<code>fasp_blas_dcsl_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)</code>	352
9.79	<code>schwarz_setup.c</code> File Reference	352
9.79.1	Detailed Description	353
9.79.2	Function Documentation	353
9.79.2.1	<code>fasp_dcsl_Schwarz_backward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)</code>	353
9.79.2.2	<code>fasp_dcsl_Schwarz_forward_smoother(Schwarz_data *Schwarz, Schwarz_param *param, dvector *x, dvector *b)</code>	353
9.79.2.3	<code>fasp_Schwarz_get_block_matrix(Schwarz_data *Schwarz, INT nblk, INT *iblock, INT *jblock, INT *mask)</code>	354
9.79.2.4	<code>fasp_Schwarz_setup(Schwarz_data *Schwarz, Schwarz_param *param)</code>	354
9.80	<code>smat.c</code> File Reference	355
9.80.1	Detailed Description	356
9.80.2	Macro Definition Documentation	356

9.80.2.1	SWAP	356
9.80.3	Function Documentation	356
9.80.3.1	fasp_blas_smat_inv(REAL *a, const INT n)	356
9.80.3.2	fasp_blas_smat_inv_nc(REAL *a, const INT n)	356
9.80.3.3	fasp_blas_smat_inv_nc2(REAL *a)	357
9.80.3.4	fasp_blas_smat_inv_nc3(REAL *a)	357
9.80.3.5	fasp_blas_smat_inv_nc4(REAL *a)	357
9.80.3.6	fasp_blas_smat_inv_nc5(REAL *a)	358
9.80.3.7	fasp_blas_smat_inv_nc7(REAL *a)	358
9.80.3.8	fasp_blas_smat_invp_nc(REAL *a, const INT n)	358
9.80.3.9	fasp_blas_smat_Linfinity(REAL *A, const INT n)	359
9.80.3.10	fasp_iden_free(idenmat *A)	359
9.80.3.11	fasp_smat_identity(REAL *a, const INT n, const INT n2)	360
9.80.3.12	fasp_smat_identity_nc2(REAL *a)	361
9.80.3.13	fasp_smat_identity_nc3(REAL *a)	361
9.80.3.14	fasp_smat_identity_nc5(REAL *a)	361
9.80.3.15	fasp_smat_identity_nc7(REAL *a)	362
9.81	smoother_bsr.c File Reference	362
9.81.1	Detailed Description	363
9.81.2	Function Documentation	363
9.81.2.1	fasp_smoother_dbsr_gs(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark)	363
9.81.2.2	fasp_smoother_dbsr_gs1(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv)	364
9.81.2.3	fasp_smoother_dbsr_gs_ascend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	364
9.81.2.4	fasp_smoother_dbsr_gs_ascend1(dBSRmat *A, dvector *b, dvector *u)	365
9.81.2.5	fasp_smoother_dbsr_gs_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	365
9.81.2.6	fasp_smoother_dbsr_gs_descend1(dBSRmat *A, dvector *b, dvector *u)	366
9.81.2.7	fasp_smoother_dbsr_gs_order1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark)	366
9.81.2.8	fasp_smoother_dbsr_gs_order2(dBSRmat *A, dvector *b, dvector *u, INT *mark, REAL *work)	366
9.81.2.9	fasp_smoother_dbsr_ilu(dBSRmat *A, dvector *b, dvector *x, void *data)	367
9.81.2.10	fasp_smoother_dbsr_jacobi(dBSRmat *A, dvector *b, dvector *u)	367
9.81.2.11	fasp_smoother_dbsr_jacobi1(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	368
9.81.2.12	fasp_smoother_dbsr_jacobi_setup(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv)	368
9.81.2.13	fasp_smoother_dbsr_sor(dBSRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL weight)	369

9.81.2.14 fasp_smoother_dbsr_sor1(dBSRmat *A, dvector *b, dvector *u, INT order, IN← T *mark, REAL *diaginv, REAL weight)	369
9.81.2.15 fasp_smoother_dbsr_sor_ascend(dBSRmat *A, dvector *b, dvector *u, REA← L *diaginv, REAL weight)	370
9.81.2.16 fasp_smoother_dbsr_sor_descend(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)	370
9.81.2.17 fasp_smoother_dbsr_sor_order(dBSRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight)	371
9.82 smoother_csr.c File Reference	371
9.82.1 Detailed Description	372
9.82.2 Function Documentation	372
9.82.2.1 fasp_smoother_dcsr_gs(dvector *u, const INT i_1, const INT i_n, const INT s, dCS← Rmat *A, dvector *b, INT L)	372
9.82.2.2 fasp_smoother_dcsr_gs_cf(dvector *u, dCSRmat *A, dvector *b, INT L, INT *mark, const INT order)	373
9.82.2.3 fasp_smoother_dcsr_gs_rb3d(dvector *u, dCSRmat *A, dvector *b, INT L, INT order, INT *mark, INT maximap, INT nx, INT ny, INT nz)	373
9.82.2.4 fasp_smoother_dcsr_ilu(dCSRmat *A, dvector *b, dvector *x, void *data)	374
9.82.2.5 fasp_smoother_dcsr_jacobi(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L)	374
9.82.2.6 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)	375
9.82.2.7 fasp_smoother_dcsr_L1diag(dvector *u, const INT i_1, const INT i_n, const INT s, dCSRmat *A, dvector *b, INT L)	375
9.82.2.8 fasp_smoother_dcsr_sgs(dvector *u, dCSRmat *A, dvector *b, INT L)	376
9.82.2.9 fasp_smoother_dcsr_sor(dvector *u, const INT i_1, const INT i_n, const INT s, dC← SRmat *A, dvector *b, INT L, const REAL w)	376
9.82.2.10 fasp_smoother_dcsr_sor_cf(dvector *u, dCSRmat *A, dvector *b, INT L, const REAL w, INT *mark, const INT order)	377
9.83 smoother_csr_cr.c File Reference	377
9.83.1 Detailed Description	378
9.83.2 Function Documentation	378
9.83.2.1 fasp_smoother_dcsr_gscr(INT pt, INT n, REAL *u, INT *ia, INT *ja, REAL *a, REAL *b, INT L, INT *CF)	378
9.84 smoother_csr_poly.c File Reference	379
9.84.1 Detailed Description	379
9.84.2 Function Documentation	379
9.84.2.1 fasp_smoother_dcsr_poly(dCSRmat *Amat, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L)	379
9.84.2.2 fasp_smoother_dcsr_poly_old(dCSRmat *Amat, dvector *brhs, dvector *usol, INT n, INT ndeg, INT L)	380
9.85 smoother_str.c File Reference	381

9.85.1 Detailed Description	382
9.85.2 Function Documentation	382
9.85.2.1 fasp_generate_diaginv_block(dSTRmat *A, ivector *neigh, dvector *diaginv, ivector *pivot)	382
9.85.2.2 fasp_smoother_dstr_gs(dSTRmat *A, dvector *b, dvector *u, INT order, INT *mark)	382
9.85.2.3 fasp_smoother_dstr_gs1(dSTRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv)	383
9.85.2.4 fasp_smoother_dstr_gs_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	383
9.85.2.5 fasp_smoother_dstr_gs_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, INT order)	384
9.85.2.6 fasp_smoother_dstr_gs_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	384
9.85.2.7 fasp_smoother_dstr_gs_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark)	385
9.85.2.8 fasp_smoother_dstr_jacobi(dSTRmat *A, dvector *b, dvector *u)	385
9.85.2.9 fasp_smoother_dstr_jacobi1(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv)	386
9.85.2.10 fasp_smoother_dstr_schwarz(dSTRmat *A, dvector *b, dvector *u, dvector *diaginv, ivector *pivot, ivector *neigh, ivector *order)	386
9.85.2.11 fasp_smoother_dstr_sor(dSTRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL weight)	386
9.85.2.12 fasp_smoother_dstr_sor1(dSTRmat *A, dvector *b, dvector *u, INT order, INT *mark, REAL *diaginv, REAL weight)	387
9.85.2.13 fasp_smoother_dstr_sor_ascend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)	387
9.85.2.14 fasp_smoother_dstr_sor_cf(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, INT order, REAL weight)	388
9.85.2.15 fasp_smoother_dstr_sor_descend(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, REAL weight)	388
9.85.2.16 fasp_smoother_dstr_sor_order(dSTRmat *A, dvector *b, dvector *u, REAL *diaginv, INT *mark, REAL weight)	389
9.86 sparse_block.c File Reference	389
9.86.1 Detailed Description	390
9.86.2 Function Documentation	390
9.86.2.1 fasp_bdcsr_free(block_dCSRmat *A)	390
9.86.2.2 fasp_dbsr_getblk(dBSRmat *A, INT *Is, INT *Js, INT m, INT n, dBSRmat *B)	390
9.86.2.3 fasp_dbsr_getblk_dcsr(dBSRmat *A)	391
9.86.2.4 fasp_dbsr_Linfinity_dcsr(dBSRmat *A)	391
9.86.2.5 fasp_dcsr_getblk(dCSRmat *A, INT *Is, INT *Js, INT m, INT n, dCSRmat *B)	392
9.87 sparse_bsr.c File Reference	392
9.87.1 Detailed Description	393
9.87.2 Function Documentation	393

9.87.2.1	<code>fasp_dbsr_alloc(INT ROW, INT COL, INT NNZ, INT nb, INT storage_manner, dBSRmat *A)</code>	393
9.87.2.2	<code>fasp_dbsr_cp(dBSRmat *A, dBSRmat *B)</code>	394
9.87.2.3	<code>fasp_dbsr_create(INT ROW, INT COL, INT NNZ, INT nb, INT storage_manner)</code>	394
9.87.2.4	<code>fasp_dbsr_diaginv(dBSRmat *A)</code>	395
9.87.2.5	<code>fasp_dbsr_diaginv2(dBSRmat *A, REAL *diaginv)</code>	395
9.87.2.6	<code>fasp_dbsr_diaginv3(dBSRmat *A, REAL *diaginv)</code>	396
9.87.2.7	<code>fasp_dbsr_diaginv4(dBSRmat *A, REAL *diaginv)</code>	397
9.87.2.8	<code>fasp_dbsr_diagLU(dBSRmat *A, REAL *DL, REAL *DU)</code>	397
9.87.2.9	<code>fasp_dbsr_diagLU2(dBSRmat *A, REAL *DL, REAL *DU)</code>	398
9.87.2.10	<code>fasp_dbsr_diagpref(dBSRmat *A)</code>	398
9.87.2.11	<code>fasp_dbsr_free(dBSRmat *A)</code>	399
9.87.2.12	<code>fasp_dbsr_getdiag(INT n, dBSRmat *A, REAL *diag)</code>	399
9.87.2.13	<code>fasp_dbsr_getdiaginv(dBSRmat *A)</code>	400
9.87.2.14	<code>fasp_dbsr_null(dBSRmat *A)</code>	400
9.87.2.15	<code>fasp_dbsr_trans(dBSRmat *A, dBSRmat *AT)</code>	401
9.88	<code>sparse_coo.c</code> File Reference	401
9.88.1	Detailed Description	402
9.88.2	Function Documentation	402
9.88.2.1	<code>fasp_dcoo_alloc(const INT m, const INT n, const INT nnz, dCOOmat *A)</code>	402
9.88.2.2	<code>fasp_dcoo_create(INT m, INT n, INT nnz)</code>	402
9.88.2.3	<code>fasp_dcoo_free(dCOOmat *A)</code>	402
9.88.2.4	<code>fasp_dcoo_shift(dCOOmat *A, INT offset)</code>	403
9.89	<code>sparse_csr.c</code> File Reference	403
9.89.1	Detailed Description	404
9.89.2	Function Documentation	405
9.89.2.1	<code>fasp_dcsr_alloc(const INT m, const INT n, const INT nnz, dCSRmat *A)</code>	405
9.89.2.2	<code>fasp_dcsr_compress(dCSRmat *A, dCSRmat *B, REAL dtol)</code>	406
9.89.2.3	<code>fasp_dcsr_compress_inplace(dCSRmat *A, REAL dtol)</code>	406
9.89.2.4	<code>fasp_dcsr_cp(dCSRmat *A, dCSRmat *B)</code>	407
9.89.2.5	<code>fasp_dcsr_create(const INT m, const INT n, const INT nnz)</code>	407
9.89.2.6	<code>fasp_dcsr_diagpref(dCSRmat *A)</code>	408
9.89.2.7	<code>fasp_dcsr_free(dCSRmat *A)</code>	409
9.89.2.8	<code>fasp_dcsr_getcol(const INT n, dCSRmat *A, REAL *col)</code>	409
9.89.2.9	<code>fasp_dcsr_getdiag(INT n, dCSRmat *A, dvector *diag)</code>	410
9.89.2.10	<code>fasp_dcsr_multicoloring(dCSRmat *A, INT *flags, INT *groups)</code>	410
9.89.2.11	<code>fasp_dcsr_null(dCSRmat *A)</code>	411

9.89.2.12	<code>fasp_dcsr_perm(dCSRmat *A, INT *P)</code>	411
9.89.2.13	<code>fasp_dcsr_regdiag(dCSRmat *A, REAL value)</code>	411
9.89.2.14	<code>fasp_dcsr_shift(dCSRmat *A, INT offset)</code>	412
9.89.2.15	<code>fasp_dcsr_sort(dCSRmat *A)</code>	412
9.89.2.16	<code>fasp_dcsr_symdiagscale(dCSRmat *A, dvector *diag)</code>	413
9.89.2.17	<code>fasp_dcsr_sympat(dCSRmat *A)</code>	414
9.89.2.18	<code>fasp_dcsr_trans(dCSRmat *A, dCSRmat *AT)</code>	414
9.89.2.19	<code>fasp_icsr_cp(iCSRmat *A, iCSRmat *B)</code>	415
9.89.2.20	<code>fasp_icsr_create(const INT m, const INT n, const INT nnz)</code>	415
9.89.2.21	<code>fasp_icsr_free(iCSRmat *A)</code>	415
9.89.2.22	<code>fasp_icsr_null(iCSRmat *A)</code>	416
9.89.2.23	<code>fasp_icsr_trans(iCSRmat *A, iCSRmat *AT)</code>	416
9.90	<code>sparse_csrl.c</code> File Reference	417
9.90.1	Detailed Description	417
9.90.2	Function Documentation	417
9.90.2.1	<code>fasp_dcsrl_create(INT num_rows, INT num_cols, INT num_nonzeros)</code>	417
9.90.2.2	<code>fasp_dcsrl_free(dCSRLmat *A)</code>	417
9.91	<code>sparse_str.c</code> File Reference	418
9.91.1	Detailed Description	418
9.91.2	Function Documentation	418
9.91.2.1	<code>fasp_dstr_alloc(INT nx, INT ny, INT nz, INT nxy, INT ngrid, INT nband, INT nc, INT *offsets, dSTRmat *A)</code>	418
9.91.2.2	<code>fasp_dstr_cp(dSTRmat *A, dSTRmat *A1)</code>	419
9.91.2.3	<code>fasp_dstr_create(INT nx, INT ny, INT nz, INT nc, INT nband, INT *offsets)</code>	419
9.91.2.4	<code>fasp_dstr_free(dSTRmat *A)</code>	420
9.91.2.5	<code>fasp_dstr_null(dSTRmat *A)</code>	420
9.92	<code>sparse_util.c</code> File Reference	421
9.92.1	Detailed Description	422
9.92.2	Function Documentation	422
9.92.2.1	<code>fasp_sparse_aat_(INT *ia, INT *ja, REAL *a, INT *na, INT *ma, INT *iat, INT *jat, REAL *at)</code>	422
9.92.2.2	<code>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)</code>	422
9.92.2.3	<code>fasp_sparse_abybms_(INT *ia, INT *ja, INT *ib, INT *jb, INT *nap, INT *map, INT *mbp, INT *ic, INT *jc)</code>	423
9.92.2.4	<code>fasp_sparse_aplbms_(INT *ia, INT *ja, INT *ib, INT *jb, INT *nab, INT *mab, INT *ic, INT *jc)</code>	423
9.92.2.5	<code>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)</code>	423

9.92.2.6	<code>fasp_sparse_iit</code> (<code>INT *ia</code> , <code>INT *ja</code> , <code>INT *na</code> , <code>INT *ma</code> , <code>INT *iat</code> , <code>INT *jat</code>)	424
9.92.2.7	<code>fasp_sparse_MIS</code> (<code>dCSRmat *A</code>)	424
9.92.2.8	<code>fasp_sparse_rapcmp</code> (<code>INT *ir</code> , <code>INT *jr</code> , <code>REAL *r</code> , <code>INT *ia</code> , <code>INT *ja</code> , <code>REAL *a</code> , <code>INT *ipt</code> , <code>INT *jpt</code> , <code>REAL *pt</code> , <code>INT *nin</code> , <code>INT *ncin</code> , <code>INT *iac</code> , <code>INT *jac</code> , <code>REAL *ac</code> , <code>INT *idummy</code>)	425
9.92.2.9	<code>fasp_sparse_rapms</code> (<code>INT *ir</code> , <code>INT *jr</code> , <code>INT *ia</code> , <code>INT *ja</code> , <code>INT *ip</code> , <code>INT *jp</code> , <code>INT *nin</code> , <code>INT</code> <code>*ncin</code> , <code>INT *iac</code> , <code>INT *jac</code> , <code>INT *maxrout</code>)	425
9.92.2.10	<code>fasp_sparse_wta</code> (<code>INT *jw</code> , <code>REAL *w</code> , <code>INT *ia</code> , <code>INT *ja</code> , <code>REAL *a</code> , <code>INT *nwp</code> , <code>INT</code> <code>*map</code> , <code>INT *jv</code> , <code>REAL *v</code> , <code>INT *nvp</code>)	426
9.92.2.11	<code>fasp_sparse_wtams</code> (<code>INT *jw</code> , <code>INT *ia</code> , <code>INT *ja</code> , <code>INT *nwp</code> , <code>INT *map</code> , <code>INT *jv</code> , <code>INT</code> <code>*nvp</code> , <code>INT *icp</code>)	426
9.92.2.12	<code>fasp_sparse_ytx</code> (<code>INT *jy</code> , <code>REAL *y</code> , <code>INT *jx</code> , <code>REAL *x</code> , <code>INT *nyp</code> , <code>INT *nxp</code> , <code>INT *icp</code> , <code>REAL *s</code>)	427
9.92.2.13	<code>fasp_sparse_ytxbig</code> (<code>INT *jy</code> , <code>REAL *y</code> , <code>INT *nyp</code> , <code>REAL *x</code> , <code>REAL *s</code>)	427
9.93	<code>spbcgs.c</code> File Reference	427
9.93.1	Detailed Description	428
9.93.2	Function Documentation	429
9.93.2.1	<code>fasp_solver_bdcsr_spbcgs</code> (<code>block_dCSRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const REAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	429
9.93.2.2	<code>fasp_solver_dbsr_spbcgs</code> (<code>dBSRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const</code> <code>REAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	430
9.93.2.3	<code>fasp_solver_dcsr_spbcgs</code> (<code>dCSRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const</code> <code>REAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	430
9.93.2.4	<code>fasp_solver_dstr_spbcgs</code> (<code>dSTRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const</code> <code>REAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	431
9.94	<code>spcg.c</code> File Reference	432
9.94.1	Detailed Description	432
9.94.2	Function Documentation	433
9.94.2.1	<code>fasp_solver_bdcsr_spcg</code> (<code>block_dCSRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const REAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	433
9.94.2.2	<code>fasp_solver_dcsr_spcg</code> (<code>dCSRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const R↵</code> <code>EAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	434
9.94.2.3	<code>fasp_solver_dstr_spcg</code> (<code>dSTRmat *A</code> , <code>dvector *b</code> , <code>dvector *u</code> , <code>precond *pc</code> , <code>const R↵</code> <code>EAL tol</code> , <code>const INT MaxIt</code> , <code>const SHORT stop_type</code> , <code>const SHORT prtlvl</code>)	434
9.95	<code>spgmres.c</code> File Reference	435
9.95.1	Detailed Description	436
9.95.2	Function Documentation	436
9.95.2.1	<code>fasp_solver_bdcsr_spgmres</code> (<code>block_dCSRmat *A</code> , <code>dvector *b</code> , <code>dvector *x</code> , <code>precond</code> <code>*pc</code> , <code>const REAL tol</code> , <code>const INT MaxIt</code> , <code>SHORT restart</code> , <code>const SHORT stop_type</code> , <code>const</code> <code>SHORT prtlvl</code>)	436
9.95.2.2	<code>fasp_solver_dbsr_spgmres</code> (<code>dBSRmat *A</code> , <code>dvector *b</code> , <code>dvector *x</code> , <code>precond *pc</code> , <code>const</code> <code>REAL tol</code> , <code>const INT MaxIt</code> , <code>SHORT restart</code> , <code>const SHORT stop_type</code> , <code>const SHORT</code> <code>prtlvl</code>)	436

9.95.2.3	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)	437
9.95.2.4	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)	438
9.96	spminres.c File Reference	438
9.96.1	Detailed Description	439
9.96.2	Function Documentation	440
9.96.2.1	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)	440
9.96.2.2	fasp_solver_dcsr_spminres(dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	440
9.96.2.3	fasp_solver_dstr_spminres(dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)	441
9.97	spvgmres.c File Reference	442
9.97.1	Detailed Description	442
9.97.2	Function Documentation	442
9.97.2.1	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)	442
9.97.2.2	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)	443
9.97.2.3	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)	444
9.97.2.4	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)	445
9.98	threads.c File Reference	446
9.98.1	Detailed Description	446
9.98.2	Function Documentation	446
9.98.2.1	FASP_GET_START_END(INT procid, INT nprocs, INT n, INT *start, INT *end)	446
9.98.2.2	fasp_set_GS_threads(INT mythreads, INT its)	447
9.98.3	Variable Documentation	448
9.98.3.1	THDs_AMG_GS	448
9.98.3.2	THDs_CPR_gGS	448
9.98.3.3	THDs_CPR_IGS	448
9.99	timing.c File Reference	448
9.99.1	Detailed Description	449
9.99.2	Function Documentation	449

9.99.2.1 fasp_gettime(REAL *time)	449
9.100vec.c File Reference	449
9.100.1 Detailed Description	450
9.100.2 Function Documentation	450
9.100.2.1 fasp_dvec_alloc(const INT m, dvector *u)	450
9.100.2.2 fasp_dvec_cp(dvector *x, dvector *y)	450
9.100.2.3 fasp_dvec_create(const INT m)	451
9.100.2.4 fasp_dvec_free(dvector *u)	451
9.100.2.5 fasp_dvec_isnan(dvector *u)	452
9.100.2.6 fasp_dvec_maxdiff(dvector *x, dvector *y)	453
9.100.2.7 fasp_dvec_null(dvector *x)	453
9.100.2.8 fasp_dvec_rand(const INT n, dvector *x)	454
9.100.2.9 fasp_dvec_set(INT n, dvector *x, REAL val)	454
9.100.2.10 fasp_dvec_symdiagscale(dvector *b, dvector *diag)	455
9.100.2.11 fasp_ivec_alloc(const INT m, ivec *u)	455
9.100.2.12 fasp_ivec_create(const INT m)	456
9.100.2.13 fasp_ivec_free(ivec *u)	457
9.100.2.14 fasp_ivec_set(const INT m, ivec *u)	457
9.101wrapper.c File Reference	458
9.101.1 Detailed Description	458
9.101.2 Function Documentation	458
9.101.2.1 fasp_fwrapper_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	458
9.101.2.2 fasp_fwrapper_krylov_amg_(INT *n, INT *nnz, INT *ia, INT *ja, REAL *a, REAL *b, REAL *u, REAL *tol, INT *maxit, INT *ptrlvl)	459
9.101.2.3 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)	459
9.101.2.4 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)	460

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

Chapter 2

How to obtain FASP

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

The most updated version of FASP can be downloaded from

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

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

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

will give you the developer version of the FASP package.

Chapter 3

Building and Installation

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

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

```
$ make config
```

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

```
$ make install
```

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

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

```
$ wish fasp_install.tcl
```

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

```
$ make help
```

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

Chapter 4

Developers

Project leader:

- Xu, Jinchao (Penn State University, USA)

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 Univeristy, USA)
- Zikatanov, Ludmil (Penn State Univeristy, USA)

With contributions from (in alphabetic order):

- Brannick, James (Penn State University, USA)
- Chen, Long (University of California, Irvine, USA)
- Huang, Feiteng (Sichuang 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)

Project coordinator:

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

Chapter 5

Doxygen

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

<http://www.doxygen.org>

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

Chapter 6

Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

AMG_data	Data for AMG solvers	19
AMG_data_bsr	Data for multigrid levels. (BSR format)	20
AMG_param	Parameters for AMG solver	22
block_BSR	Block REAL matrix format for reservoir simulation	24
block_dCSRmat	Block REAL CSR matrix format	24
block_dvector	Block REAL vector structure	25
block_iCSRmat	Block INT CSR matrix format	25
block_ivector	Block INT vector structure	26
block_Reservoir	Block REAL matrix format for reservoir simulation	27
dBSRmat	Block sparse row storage matrix of REAL type	27
dCOOmat	Sparse matrix of REAL type in COO (or IJ) format	28
dCSRLmat	Sparse matrix of REAL type in CSRL format	29
dCSRmat	Sparse matrix of REAL type in CSR format	30
ddenmat	Dense matrix of REAL type	30
dSTRmat	Structure matrix of REAL type	31
dvector	Vector with n entries of REAL type	32
grid2d	Two dimensional grid data structure	32

iCOOmat	Sparse matrix of INT type in COO (or IJ) format	34
iCSRmat	Sparse matrix of INT type in CSR format	35
idenmat	Dense matrix of INT type	36
ILU_data	Data for ILU setup	36
ILU_param	Parameters for ILU	37
input_param	Input parameters	38
itsolver_param	Parameters passed to iterative solvers	45
ivector	Vector with n entries of INT type	46
Link	Struct for Links	47
linked_list	A linked list node	47
Mumps_data	Parameters for MUMPS interface	48
mxv_matfree	Matrix-vector multiplication, replace the actual matrix	48
precond	Preconditioner data and action	49
precond_block_data	Data passed to the preconditioner for block preconditioning for block_dCSRmat format	49
precond_block_reservoir_data	Data passed to the preconditioner for preconditioning reservoir simulation problems	51
precond_data	Data passed to the preconditioners	54
precond_data_bsr	Data passed to the preconditioners	55
precond_data_str	Data passed to the preconditioner for dSTRmat matrices	57
precond_diagbsr	Data passed to diagonal preconditioner for dBSRmat matrices	59
precond_diagstr	Data passed to diagonal preconditioner for dSTRmat matrices	59
precond_FASP_blkoi_data	Data passed to the preconditioner for preconditioning reservoir simulation problems	60
precond_sweeping_data	Data passed to the preconditioner for sweeping preconditioning	64
Schwarz_data	Data for Schwarz methods	66
Schwarz_param	Parameters for Schwarz method	67

Chapter 7

File Index

7.1 File List

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

amg.c	AMG method as an iterative solver (main file)	69
amg_setup_cr.c	Brannick-Falgout compatible relaxation based AMG: SETUP phase	70
amg_setup_rs.c	Ruge-Stuben AMG: SETUP phase	71
amg_setup_sa.c	Smoothed aggregation AMG: SETUP phase	72
amg_setup_ua.c	Unsmoothed aggregation AMG: SETUP phase	74
amg_solve.c	Algebraic multigrid iterations: SOLVE phase	76
amlirecur.c	Abstract AMLI multilevel iteration – recursive version	79
array.c	Simple array operations – init, set, copy, etc	82
blas_array.c	BLAS operations for arrays	86
blas_bcsr.c	BLAS operations for block_dCSRmat matrices	91
blas_bsr.c	BLAS operations for dBSRmat matrices	93
blas_csr.c	BLAS operations for dCSRmat matrices	100
blas_csrl.c	BLAS operations for dCSRLmat matrices	108
blas_smat.c	BLAS operations for <i>small</i> dense matrix	109
blas_str.c	BLAS operations for dSTRmat matrices	130
blas_vec.c	BLAS operations for vectors	132
checkmat.c	Check matrix properties	136

coarsening_cr.c	
Coarsening with Brannick-Falgout strategy	141
coarsening_rs.c	
Coarsening with a modified Ruge-Stuben strategy	142
convert.c	
Some utilities for format conversion	144
doxygen.h	
Main page for Doxygen documentation	148
eigen.c	
Simple subroutines for compute the extreme eigenvalues	148
factor.f	
LU factorization for CSR matrix	149
famg.c	
Full AMG method as an iterative solver (main file)	149
fasp.h	
Main header file for FASP	150
fasp_block.h	
Main header file for FASP (block matrices)	158
fasp_const.h	
Definition of all kinds of messages, including error messages, solver types, etc	160
fmgcycle.c	
Abstract non-recursive full multigrid cycle	180
formats.c	
Matrix format conversion routines	181
givens.c	
Givens transformation	187
gmg_poisson.c	
GMG method as an iterative solver for Poisson Problem	188
graphics.c	
Functions for graphical output	194
ilu.f	
ILU routines for preconditioning adapted from SPARSEKIT	198
ilu_setup_bsr.c	
Setup Incomplete LU decomposition for dBSRmat matrices	198
ilu_setup_csr.c	
Setup of ILU decomposition for dCSRmat matrices	199
ilu_setup_str.c	
Setup of ILU decomposition for dSTRmat matrices	200
init.c	
Initialize important data structures	202
input.c	
Read input parameters	209
interface_mumps.c	
Interface to MUMPS direct solvers	210
interface_samg.c	
Interface to SAMG	212
interface_superlu.c	
Interface to SuperLU direct solvers	214
interface_umfpack.c	
Interface to UMFPACK direct solvers	214
interpolation.c	
Interpolation operators for AMG	215
interpolation_em.c	
Interpolation operators for AMG based on energy-min	217

io.c	Matrix-vector input/output subroutines	218
itsolver_bcsr.c	Iterative solvers for block_dCSRmat matrices	241
itsolver_bsr.c	Iterative solvers for dBSRmat matrices	245
itsolver_csr.c	Iterative solvers for dCSRmat matrices	250
itsolver_mf.c	Iterative solvers with matrix-free spmv	255
itsolver_str.c	Iterative solvers for dSTRmat matrices	257
lu.c	LU decomposition and direct solve for dense matrix	260
memory.c	Memory allocation and deallocation	262
message.c	Output some useful messages	266
mgcycle.c	Abstract non-recursive multigrid cycle	269
mgrecur.c	Abstract multigrid cycle – recursive version	271
ordering.c	A collection of ordering, merging, removing duplicated integers functions	272
parameters.c	Initialize, set, or print input data and parameters	277
pbcgs.c	Krylov subspace methods – Preconditioned BiCGstab	286
pbcgs_mf.c	Krylov subspace methods – Preconditioned BiCGstab (matrix free)	290
pcg.c	Krylov subspace methods – Preconditioned conjugate gradient	292
pcg_mf.c	Krylov subspace methods – Preconditioned conjugate gradient (matrix free)	298
pgcg.c	Krylov subspace methods – Preconditioned Generalized CG	300
pgcg_mf.c	Krylov subspace methods – Preconditioned Generalized CG (matrix free)	301
pgcr.c	Krylov subspace methods – Preconditioned GCR	302
pgmres.c	Krylov subspace methods – Right-preconditioned GMRes	305
pgmres_mf.c	Krylov subspace methods – Preconditioned GMRes (matrix free)	308
pminres.c	Krylov subspace methods – Preconditioned minimal residual	309
pminres_mf.c	Krylov subspace methods – Preconditioned minimal residual (matrix free)	313
precond_bcsr.c	Preconditioners for block CSR matrices	315
precond_bsr.c	Preconditioners for dBSRmat matrices	323
precond_csr.c	Preconditioners for dCSRmat matrices	330

precond_str.c	Preconditioners for dSTRmat matrices	336
pvfgmres.c	Preconditioned variable-restarting flexible GMRes	341
pvfgmres_mf.c	Krylov subspace methods – Preconditioned variable-restarting flexible GMRes (matrix free)	343
pvgmres.c	Krylov subspace methods – Preconditioned variable-restart GMRes	345
pvgmres_mf.c	Krylov subspace methods – Preconditioned variable-restarting GMRes (matrix free)	349
quadrature.c	Quadrature rules	350
rap.c	Tripple-matrix multiplication $R \cdot A \cdot P$	351
schwarz_setup.c	Setup phase for the Schwarz methods	352
smat.c	Simple operations for <i>small</i> dense matrices in row-major format	355
smoother_bsr.c	Smoothers for dBSRmat matrices	362
smoother_csr.c	Smoothers for dCSRmat matrices	371
smoother_csr_cr.c	Smoothers for dCSRmat matrices using compatible relaxation	377
smoother_csr_poly.c	Smoothers for dCSRmat matrices using poly. approx. to A^{-1}	379
smoother_str.c	Smoothers for dSTRmat matrices	381
sparse_block.c	Sparse matrix block operations	389
sparse_bsr.c	Sparse matrix operations for dBSRmat matrices	392
sparse_coo.c	Sparse matrix operations for dCOOmat matrices	401
sparse_csr.c	Sparse matrix operations for dCSRmat matrices	403
sparse_csrl.c	Sparse matrix operations for dCSRLmat matrices	417
sparse_str.c	Sparse matrix operations for dSTRmat matrices	418
sparse_util.c	Routines for sparse matrix operations	421
spbcgs.c	Krylov subspace methods – Preconditioned BiCGstab with safe net	427
spcg.c	Krylov subspace methods – Preconditioned conjugate gradient with safe net	432
spgmres.c	Krylov subspace methods – Preconditioned GMRes with safe net	435
spminres.c	Krylov subspace methods – Preconditioned minimal residual with safe net	438
spvgmres.c	Krylov subspace methods – Preconditioned variable-restart GMRes with safe net	442
threads.c	Get and set number of threads and assign work load for each thread	446

timing.c	Timing subroutines	448
vec.c	Simple operations for vectors	449
wrapper.c	Wrappers for accessing functions by advanced users	458

Chapter 8

Data Structure Documentation

8.1 AMG_data Struct Reference

Data for AMG solvers.

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

Data Fields

- [SHORT max_levels](#)
max number of levels
- [SHORT num_levels](#)
number of levels in use \leq max_levels
- [dCSRmat A](#)
pointer to the matrix at level level_num
- [dCSRmat R](#)
restriction operator at level level_num
- [dCSRmat P](#)
prolongation operator at level level_num
- [dvector b](#)
pointer to the right-hand side at level level_num
- [dvector x](#)
pointer to the iterative solution at level level_num
- `void *` [Numeric](#)
pointer to the numerical factorization from UMFPACK
- [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*

8.1.1 Detailed Description

Data for AMG solvers.

Note

This is needed for the AMG solver/preconditioner.

Definition at line 684 of file fasp.h.

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

- [fasp.h](#)

8.2 AMG_data_bsr Struct Reference

Data for multigrid levels. (BSR format)

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

Data Fields

- [INT max_levels](#)
 - max number of levels*
- [INT num_levels](#)
 - number of levels in use <= 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 diainv](#)
pointer to the diagonal inverse at level level_num
- [dCSRmat Ac](#)
pointer to the matrix at level level_num (csr format)
- `void * Numeric`
pointer to the numerical dactorization from UMFPACK
- [dCSRmat PP](#)
pointer to the pressure block (only for reservoir simulation)
- `REAL * pw`
pointer to the auxiliary vectors for pressure block
- [dBSRmat SS](#)
pointer to the saturation block (only for reservoir simulation)
- `REAL * sw`
pointer to the auxiliary vectors for saturation block
- [dvector diainv_SS](#)
pointer to the diagonal inverse of the saturation block at level level_num
- [ILU_data PP_LU](#)
ILU data for pressure block.
- [ivector cfmark](#)
pointer to the CF marker at level level_num
- [INT ILU_levels](#)
number of levels use ILU smoother
- [ILU_data LU](#)
ILU matrix for ILU smoother.
- [INT near_kernel_dim](#)
dimension of the near kernel for SAMG
- `REAL ** near_kernel_basis`
basis of near kernel space for SAMG
- `dCSRmat * A_nk`
Matrix data for near kernal.
- `dCSRmat * P_nk`
Prolongation for near kernal.
- `dCSRmat * R_nk`
Resriction for near kernal.
- [dvector w](#)
temporary work space
- [Mumps_data mumps](#)
data for MUMPS

8.2.1 Detailed Description

Data for multigrid levels. (BSR format)

Note

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

Definition at line 191 of file fasp_block.h.

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

- [fasp_block.h](#)

8.3 AMG_param Struct Reference

Parameters for AMG solver.

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

Data Fields

- [SHORT AMG_type](#)
type of AMG method
- [SHORT print_level](#)
print level for AMG
- [INT maxit](#)
max number of iterations of AMG
- [REAL tol](#)
stopping tolerance for AMG solver
- [SHORT max_levels](#)
max number of levels of AMG
- [INT coarse_dof](#)
max number of coarsest level DOF
- [SHORT cycle_type](#)
type of AMG cycle
- [REAL quality_bound](#)
quality threshold for pairwise aggregation
- [SHORT smoother](#)
smoother type
- [SHORT smooth_order](#)
smoother order
- [SHORT presmooth_iter](#)
number of 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_lfil](#)
level of fill-in for ILUs and ILUK
- [REAL ILU_droptol](#)
drop tolerance for ILUt
- [REAL ILU_relax](#)
relaxation for ILUs
- [REAL ILU_permtol](#)
*permuted if $\text{permtol} * |a(i,j)| > |a(i,i)|$*
- [INT Schwarz_levels](#)
number of levels use Schwarz smoother
- [INT Schwarz_mmsize](#)
maximal block size
- [INT Schwarz_maxlvl](#)
maximal levels
- [INT Schwarz_type](#)
type of Schwarz method
- [INT Schwarz_blksolver](#)
type of Schwarz block solver

8.3.1 Detailed Description

Parameters for AMG solver.

Note

This is needed for the AMG solver/preconditioner.

Definition at line 545 of file fasp.h.

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

- [fasp.h](#)

8.4 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

8.4.1 Detailed Description

Block REAL matrix format for reservoir simulation.

Definition at line 165 of file fasp_block.h.

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

- [fasp_block.h](#)

8.5 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]

8.5.1 Detailed Description

Block REAL CSR matrix format.

Note

The starting index of A is 0.

Definition at line 77 of file fasp_block.h.

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

- [fasp_block.h](#)

8.6 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]

8.6.1 Detailed Description

Block REAL vector structure.

Definition at line 113 of file fasp_block.h.

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

- [fasp_block.h](#)

8.7 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]

8.7.1 Detailed Description

Block INT CSR matrix format.

Note

The starting index of A is 0.

Definition at line 96 of file fasp_block.h.

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

- [fasp_block.h](#)

8.8 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]

8.8.1 Detailed Description

Block INT vector structure.

Note

The starting index of A is 0.

Definition at line 129 of file fasp_block.h.

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

- [fasp_block.h](#)

8.9 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

8.9.1 Detailed Description

Block REAL matrix format for reservoir simulation.

Definition at line 144 of file fasp_block.h.

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

- [fasp_block.h](#)

8.10 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](#)

8.10.1 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 37 of file fasp_block.h.

8.10.2 Field Documentation

8.10.2.1 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 67 of file fasp_block.h.

8.10.2.2 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 60 of file fasp_block.h.

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

- [fasp_block.h](#)

8.11 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

8.11.1 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 199 of file fasp.h.

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

- [fasp.h](#)

8.12 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

8.12.1 Detailed Description

Sparse matrix of REAL type in CSRL format.

Definition at line 255 of file fasp.h.

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

- [fasp.h](#)

8.13 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

8.13.1 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 138 of file fasp.h.

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

- [fasp.h](#)

8.14 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

8.14.1 Detailed Description

Dense matrix of REAL type.

A dense REAL matrix

Definition at line 98 of file fasp.h.

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

- [fasp.h](#)

8.15 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])*

8.15.1 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 294 of file `fasp.h`.

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

- [fasp.h](#)

8.16 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

8.16.1 Detailed Description

Vector with `n` entries of REAL type.

Definition at line 332 of file `fasp.h`.

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

- [fasp.h](#)

8.17 grid2d Struct Reference

Two dimensional grid data structure.

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

Data Fields

- [REAL\(* p\)\[2\]](#)
- [INT\(* e\)\[2\]](#)
- [INT\(* t\)\[3\]](#)
- [INT\(* s\)\[3\]](#)

- `INT * pdir`
- `INT * edir`
- `INT * pfather`
- `INT * efather`
- `INT * tfather`
- `INT vertices`
- `INT edges`
- `INT triangles`

8.17.1 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 1086 of file `fasp.h`.

8.17.2 Field Documentation

8.17.2.1 `INT(* e)[2]`

Vertices of edges

Definition at line 1089 of file `fasp.h`.

8.17.2.2 `INT edges`

Number of edges

Definition at line 1100 of file `fasp.h`.

8.17.2.3 `INT* edir`

Boundary flags (0 \leq interior edge)

Definition at line 1093 of file `fasp.h`.

8.17.2.4 `INT* efather`

Father edge or triangle

Definition at line 1096 of file `fasp.h`.

8.17.2.5 `REAL(* p)[2]`

Coordinates of vertices

Definition at line 1088 of file `fasp.h`.

8.17.2.6 `INT* pdiri`

Boundary flags (0 <=> interior point)

Definition at line 1092 of file fasp.h.

8.17.2.7 `INT* pfather`

Father point or edge

Definition at line 1095 of file fasp.h.

8.17.2.8 `INT(* s)[3]`

Edges of triangles

Definition at line 1091 of file fasp.h.

8.17.2.9 `INT(* t)[3]`

Vertices of triangles

Definition at line 1090 of file fasp.h.

8.17.2.10 `INT* tfather`

Father triangle

Definition at line 1097 of file fasp.h.

8.17.2.11 `INT triangles`

Number of triangles

Definition at line 1101 of file fasp.h.

8.17.2.12 `INT vertices`

Number of grid points

Definition at line 1099 of file fasp.h.

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

- [fasp.h](#)

8.18 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

8.18.1 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 229 of file fasp.h.

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

- [fasp.h](#)

8.19 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

8.19.1 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 168 of file fasp.h.

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

- [fasp.h](#)

8.20 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

8.20.1 Detailed Description

Dense matrix of INT type.

A dense INT matrix

Definition at line 117 of file fasp.h.

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

- [fasp.h](#)

8.21 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

8.21.1 Detailed Description

Data for ILU setup.

Definition at line 390 of file fasp.h.

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

- [fasp.h](#)

8.22 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_ifil](#)
level of fill-in for ILUk
- [REAL ILU_droptol](#)
drop tolerance for ILUt
- [REAL ILU_relax](#)
add the sum of dropped elements to diagonal element in proportion relax
- [REAL ILU_permtol](#)
*permuted if $\text{permtol} * |a(i,j)| > |a(i,i)|$*

8.22.1 Detailed Description

Parameters for ILU.

Definition at line 364 of file fasp.h.

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

- [fasp.h](#)

8.23 input_param Struct Reference

Input parameters.

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

Data Fields

- [SHORT print_level](#)
- [SHORT output_type](#)
- [char inifile \[256\]](#)
- [char workdir \[256\]](#)
- [INT problem_num](#)
- [SHORT solver_type](#)
- [SHORT precondition_type](#)
- [SHORT stop_type](#)
- [REAL itsolver_tol](#)
- [INT itsolver_maxit](#)
- [INT restart](#)
- [SHORT ILU_type](#)
- [INT ILU_lfil](#)
- [REAL ILU_droptol](#)
- [REAL ILU_relax](#)
- [REAL ILU_permtol](#)
- [INT Schwarz_mmsize](#)
- [INT Schwarz_maxlvl](#)
- [INT Schwarz_type](#)
- [INT Schwarz_blksolver](#)
- [SHORT AMG_type](#)
- [SHORT AMG_levels](#)
- [SHORT AMG_cycle_type](#)
- [SHORT AMG_smoother](#)
- [SHORT AMG_smooth_order](#)
- [REAL AMG_relaxation](#)
- [SHORT AMG_polynomial_degree](#)
- [SHORT AMG_presmooth_iter](#)
- [SHORT AMG_postsMOOTH_iter](#)
- [INT AMG_coarse_dof](#)
- [REAL AMG_tol](#)
- [INT AMG_maxit](#)
- [SHORT AMG_ILU_levels](#)

- [SHORT AMG_coarse_solver](#)
- [SHORT AMG_coarse_scaling](#)
- [SHORT AMG_amli_degree](#)
- [SHORT AMG_nl_amli_krylov_type](#)
- [INT AMG_Schwarz_levels](#)
- [SHORT AMG_coarsening_type](#)
- [SHORT AMG_aggregation_type](#)
- [SHORT AMG_interpolation_type](#)
- [REAL AMG_strong_threshold](#)
- [REAL AMG_truncation_threshold](#)
- [REAL AMG_max_row_sum](#)
- [INT AMG_aggressive_level](#)
- [INT AMG_aggressive_path](#)
- [INT AMG_pair_number](#)
- [REAL AMG_quality_bound](#)
- [REAL AMG_strong_coupled](#)
- [INT AMG_max_aggregation](#)
- [REAL AMG_tentative_smooth](#)
- [SHORT AMG_smooth_filter](#)

8.23.1 Detailed Description

Input parameters.

Input parameters, reading from disk file

Definition at line 988 of file fasp.h.

8.23.2 Field Documentation

8.23.2.1 SHORT AMG_aggregation_type

aggregation type

Definition at line 1042 of file fasp.h.

8.23.2.2 INT AMG_aggressive_level

number of levels use aggressive coarsening

Definition at line 1047 of file fasp.h.

8.23.2.3 INT AMG_aggressive_path

number of paths used to determine strongly coupled C-set

Definition at line 1048 of file fasp.h.

8.23.2.4 SHORT AMG_amli_degree

degree of the polynomial used by AMLI cycle

Definition at line 1036 of file fasp.h.

8.23.2.5 INT AMG_coarse_dof

max number of coarsest level DOF

Definition at line 1030 of file fasp.h.

8.23.2.6 SHORT AMG_coarse_scaling

switch of scaling of the coarse grid correction

Definition at line 1035 of file fasp.h.

8.23.2.7 SHORT AMG_coarse_solver

coarse solver type

Definition at line 1034 of file fasp.h.

8.23.2.8 SHORT AMG_coarsening_type

coarsening type

Definition at line 1041 of file fasp.h.

8.23.2.9 SHORT AMG_cycle_type

type of cycle

Definition at line 1023 of file fasp.h.

8.23.2.10 SHORT AMG_ILU_levels

how many levels use ILU smoother

Definition at line 1033 of file fasp.h.

8.23.2.11 SHORT AMG_interpolation_type

interpolation type

Definition at line 1043 of file fasp.h.

8.23.2.12 SHORT AMG_levels

maximal number of levels

Definition at line 1022 of file fasp.h.

8.23.2.13 INT AMG_max_aggregation

max size of each aggregate

Definition at line 1054 of file fasp.h.

8.23.2.14 REAL AMG_max_row_sum

maximal row sum

Definition at line 1046 of file fasp.h.

8.23.2.15 INT AMG_maxit

number of iterations for AMG used as preconditioner

Definition at line 1032 of file fasp.h.

8.23.2.16 SHORT AMG_nl_amli_krylov_type

type of Krylov method used by nonlinear AMLI cycle

Definition at line 1037 of file fasp.h.

8.23.2.17 INT AMG_pair_number

number of pairs in matching algorithm

Definition at line 1049 of file fasp.h.

8.23.2.18 SHORT AMG_polynomial_degree

degree of the polynomial smoother

Definition at line 1027 of file fasp.h.

8.23.2.19 SHORT AMG_postsmooth_iter

number of postsmoothing

Definition at line 1029 of file fasp.h.

8.23.2.20 SHORT AMG_presmooth_iter

number of presmoothing

Definition at line 1028 of file fasp.h.

8.23.2.21 REAL AMG_quality_bound

threshold for pair wise aggregation

Definition at line 1050 of file fasp.h.

8.23.2.22 REAL AMG_relaxation

over-relaxation parameter for SOR

Definition at line 1026 of file fasp.h.

8.23.2.23 INT AMG_Schwarz_levels

number of levels use Schwarz smoother

Definition at line 1038 of file fasp.h.

8.23.2.24 SHORT AMG_smooth_filter

use filter for smoothing the tentative prolongation or not

Definition at line 1056 of file fasp.h.

8.23.2.25 SHORT AMG_smooth_order

order for smoothers

Definition at line 1025 of file fasp.h.

8.23.2.26 SHORT AMG_smoother

type of smoother

Definition at line 1024 of file fasp.h.

8.23.2.27 REAL AMG_strong_coupled

strong coupled threshold for aggregate

Definition at line 1053 of file fasp.h.

8.23.2.28 REAL AMG_strong_threshold

strong threshold for coarsening

Definition at line 1044 of file fasp.h.

8.23.2.29 REAL AMG_tentative_smooth

relaxation factor for smoothing the tentative prolongation

Definition at line 1055 of file fasp.h.

8.23.2.30 REAL AMG_tol

tolerance for AMG if used as preconditioner

Definition at line 1031 of file fasp.h.

8.23.2.31 REAL AMG_truncation_threshold

truncation factor for interpolation

Definition at line 1045 of file fasp.h.

8.23.2.32 SHORT AMG_type

Type of AMG

Definition at line 1021 of file fasp.h.

8.23.2.33 REAL ILU_droptol

drop tolerance

Definition at line 1010 of file fasp.h.

8.23.2.34 INT ILU_lfil

level of fill-in

Definition at line 1009 of file fasp.h.

8.23.2.35 REAL ILU_permtol

permutation tolerance

Definition at line 1012 of file fasp.h.

8.23.2.36 REAL ILU_relax

scaling factor: add the sum of dropped entries to diagonal

Definition at line 1011 of file fasp.h.

8.23.2.37 SHORT ILU_type

ILU type for decomposition

Definition at line 1008 of file fasp.h.

8.23.2.38 char inifile[256]

ini file name

Definition at line 995 of file fasp.h.

8.23.2.39 INT itsolver_maxit

maximal number of iterations for iterative solvers

Definition at line 1004 of file fasp.h.

8.23.2.40 REAL itsolver_tol

tolerance for iterative linear solver

Definition at line 1003 of file fasp.h.

8.23.2.41 SHORT output_type

type of output stream

Definition at line 992 of file fasp.h.

8.23.2.42 SHORT precondition_type

type of preconditioner for iterative solvers

Definition at line 1001 of file fasp.h.

8.23.2.43 SHORT print_level

print level

Definition at line 991 of file fasp.h.

8.23.2.44 INT problem_num

problem number to solve

Definition at line 997 of file fasp.h.

8.23.2.45 INT restart

restart number used in GMRES

Definition at line 1005 of file fasp.h.

8.23.2.46 INT Schwarz_blksolver

type of Schwarz block solver

Definition at line 1018 of file fasp.h.

8.23.2.47 INT Schwarz_maxlvl

maximal levels

Definition at line 1016 of file fasp.h.

8.23.2.48 INT Schwarz_mmsize

maximal block size

Definition at line 1015 of file fasp.h.

8.23.2.49 INT Schwarz_type

type of Schwarz method

Definition at line 1017 of file fasp.h.

8.23.2.50 **SHORT** solver_type

type of iterative solvers

Definition at line 1000 of file fasp.h.

8.23.2.51 **SHORT** stop_type

type of stopping criteria for iterative solvers

Definition at line 1002 of file fasp.h.

8.23.2.52 **char** workdir[256]

working directory for data files

Definition at line 996 of file fasp.h.

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

- [fasp.h](#)

8.24 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](#)

8.24.1 Detailed Description

Parameters passed to iterative solvers.

Definition at line 1064 of file fasp.h.

8.24.2 Field Documentation

8.24.2.1 **SHORT** itsolver_type

solver type: see message.h

Definition at line 1066 of file fasp.h.

8.24.2.2 INT maxit

max number of iterations

Definition at line 1069 of file fasp.h.

8.24.2.3 SHORT precond_type

preconditioner type: see message.h

Definition at line 1067 of file fasp.h.

8.24.2.4 SHORT print_level

print level: 0–10

Definition at line 1072 of file fasp.h.

8.24.2.5 INT restart

number of steps for restarting: for GMRES etc

Definition at line 1071 of file fasp.h.

8.24.2.6 SHORT stop_type

stopping criteria type

Definition at line 1068 of file fasp.h.

8.24.2.7 REAL tol

convergence tolerance

Definition at line 1070 of file fasp.h.

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

- [fasp.h](#)

8.25 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

8.25.1 Detailed Description

Vector with n entries of INT type.

Definition at line 346 of file fasp.h.

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

- [fasp.h](#)

8.26 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

8.26.1 Detailed Description

Struct for Links.

Definition at line 1113 of file fasp.h.

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

- [fasp.h](#)

8.27 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

8.27.1 Detailed Description

A linked list node.

Note

This definition is adapted from hypre 2.0.

Definition at line 1130 of file fasp.h.

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

- [fasp.h](#)

8.28 Mumps_data Struct Reference

Parameters for MUMPS interface.

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

Data Fields

- [INT job](#)
work for MUMPS

8.28.1 Detailed Description

Parameters for MUMPS interface.

Added on 10/10/2014

Definition at line 449 of file fasp.h.

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

- [fasp.h](#)

8.29 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

8.29.1 Detailed Description

Matrix-vector multiplication, replace the actual matrix.

Definition at line 972 of file fasp.h.

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

- [fasp.h](#)

8.30 precondition Struct Reference

Preconditioner data and action.

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

Data Fields

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

8.30.1 Detailed Description

Preconditioner data and action.

Note

This is the preconditioner structure for preconditioned iterative methods.

Definition at line 958 of file fasp.h.

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

- [fasp.h](#)

8.31 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](#)

8.31.1 Detailed Description

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

Data passed to the preconditioner for block diagonal preconditioning.

This is needed for the block preconditioner.

Note

This is needed for the diagonal block preconditioner.

Definition at line 492 of file [fasp_block.h](#).

8.31.2 Field Documentation

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

data for each diagonal block which need to solve in the block preconditioners

Definition at line 499 of file [fasp_block.h](#).

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

problem data, the blocks

Definition at line 497 of file [fasp_block.h](#).

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

parameters for AMG

Definition at line 511 of file [fasp_block.h](#).

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

LU decomposition for the diagonal blocks – (only for UMFpack – Xiaozhe Hu)

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

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

AMG data for the diagonal blocks

Definition at line 510 of file [fasp_block.h](#).

8.31.2.6 [dvector](#) [r](#)

temp work space

Definition at line 501 of file [fasp_block.h](#).

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

- [fasp_block.h](#)

8.32 precondition_block_reservoir_data Struct Reference

Data passed to the preconditioner for preconditioning 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 $(-I - A_{wr} * Arr^{-1} * Arw)^{-1}$, *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**

8.32.1 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 394 of file fasp_block.h.

8.32.2 Field Documentation

8.32.2.1 **precond_diagstr*** **diag**

the diagonal inverse for diagonal scaling

Definition at line 474 of file fasp_block.h.

8.32.2.2 **dvector*** **diaginv**

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

Definition at line 475 of file fasp_block.h.

8.32.2.3 **dvector*** **diaginvS**

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

Definition at line 477 of file fasp_block.h.

8.32.2.4 ivector* order

order for smoothing

Definition at line 479 of file fasp_block.h.

8.32.2.5 ivector* perf_idx

variable index for perf

Definition at line 467 of file fasp_block.h.

8.32.2.6 ivector* pivot

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

Definition at line 476 of file fasp_block.h.

8.32.2.7 ivector* pivotS

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

Definition at line 478 of file fasp_block.h.

8.32.2.8 dCSRmat* PP

pressure block after diagonal scaling

Definition at line 471 of file fasp_block.h.

8.32.2.9 dvector r

temporary dvector used to store and restore the residual

Definition at line 482 of file fasp_block.h.

8.32.2.10 dSTRmat* RR

Diagonal scaled reservoir block

Definition at line 469 of file fasp_block.h.

8.32.2.11 SHORT scaled

whether the matrix is scaled

Definition at line 466 of file fasp_block.h.

8.32.2.12 dSTRmat* SS

saturation block after diagonal scaling

Definition at line 472 of file fasp_block.h.

8.32.2.13 REAL* w

temporary work space for other usage

Definition at line 483 of file fasp_block.h.

8.32.2.14 dCSRmat* WW

Argumented well block

Definition at line 470 of file fasp_block.h.

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

- [fasp_block.h](#)

8.33 precondition_data Struct Reference

Data passed to the preconditioners.

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

Data Fields

- [SHORT AMG_type](#)
type of AMG method
- [SHORT print_level](#)
print level in AMG preconditioner
- [INT maxit](#)
max number of iterations of AMG preconditioner
- [SHORT max_levels](#)
max number of AMG levels
- [REAL tol](#)
tolerance for AMG preconditioner
- [SHORT cycle_type](#)
AMG cycle type.
- [SHORT smoother](#)
AMG smoother type.
- [SHORT smooth_order](#)
AMG smoother ordering.
- [SHORT presmooth_iter](#)
number of presmoothing
- [SHORT postsmooth_iter](#)
number of postsmoothing
- [REAL relaxation](#)
relaxation parameter for SOR smoother
- [SHORT polynomial_degree](#)
degree of the polynomial smoother
- [SHORT coarsening_type](#)

- switch of scaling of the coarse grid correction*
- [SHORT coarse_solver](#)
 - coarse solver type for AMG*
- [SHORT coarse_scaling](#)
 - switch of scaling of the coarse grid correction*
- [SHORT amli_degree](#)
 - degree of the polynomial used by AMLI cycle*
- [SHORT nl_amli_krylov_type](#)
 - type of Krylov method used by Nonlinear AMLI cycle*
- [REAL tentative_smooth](#)
 - smooth factor for smoothing the tentative prolongation*
- [REAL * amli_coef](#)
 - coefficients of the polynomial used by AMLI cycle*
- [AMG_data * mgl_data](#)
 - AMG preconditioner data.*
- [ILU_data * LU](#)
 - ILU preconditioner data (needed for CPR type preconditioner)*
- [dCSRmat * A](#)
 - Matrix data.*
- [dCSRmat * A_nk](#)
 - Matrix data for near kernel.*
- [dCSRmat * P_nk](#)
 - Prolongation for near kernel.*
- [dCSRmat * R_nk](#)
 - Restriction for near kernel.*
- [dvector r](#)
 - temporary dvector used to store and restore the residual*
- [REAL * w](#)
 - temporary work space for other usage*

8.33.1 Detailed Description

Data passed to the preconditioners.

Definition at line 754 of file fasp.h.

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

- [fasp.h](#)

8.34 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*

8.34.1 Detailed Description

Data passed to the preconditioners.

Note

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

Definition at line 301 of file fasp_block.h.

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

- [fasp_block.h](#)

8.35 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

8.35.1 Detailed Description

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

Definition at line 850 of file fasp.h.

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

- [fasp.h](#)

8.36 precondition_diagbsr Struct Reference

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

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

Data Fields

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

8.36.1 Detailed Description

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

Note

This is needed for the diagonal preconditioner.

Definition at line 283 of file `fasp_block.h`.

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

- [fasp_block.h](#)

8.37 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

8.37.1 Detailed Description

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

Note

This is needed for the diagonal preconditioner.

Definition at line 942 of file `fasp.h`.

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

- [fasp.h](#)

8.38 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 $(-I - A_{wr} * Arr^{-1} * A_{rw})^{-1}$, Arr may be replaced by LU*
- [INT](#) `maxit`
- [INT](#) `restart`
- [REAL](#) `tol`
- [dvector](#) `r`
- [REAL](#) * `w`

8.38.1 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 545 of file `fasp_block.h`.

8.38.2 Field Documentation

8.38.2.1 `block_BSR`* `A`

Part 1: Basic data.

whole jacobian system in `block_BSRmat`

Definition at line 550 of file `fasp_block.h`.

8.38.2.2 `dvector`* `diaginv`

inverse of the diagonal blocks of reservoir block

Definition at line 625 of file `fasp_block.h`.

8.38.2.3 dvector* diaginv_noscale

inverse of diagonal blocks for diagonal scaling

Definition at line 557 of file fasp_block.h.

8.38.2.4 dvector* diaginv_S

inverse of the diagonal blocks of saturation block

Definition at line 566 of file fasp_block.h.

8.38.2.5 INT maxit

max number of iterations

Definition at line 643 of file fasp_block.h.

8.38.2.6 AMG_data* mgl_data

AMG data for presure-presure block

Definition at line 571 of file fasp_block.h.

8.38.2.7 ivector* neigh

neighbor information of the reservoir block

Definition at line 561 of file fasp_block.h.

8.38.2.8 ivector* order

ordering of the reservoir block

Definition at line 562 of file fasp_block.h.

8.38.2.9 ivector* perf_idx

index of blocks which have perforation

Definition at line 632 of file fasp_block.h.

8.38.2.10 ivector* perf_neigh

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

Definition at line 633 of file fasp_block.h.

8.38.2.11 ivector* pivot

pivot for the GS smoothers for the reservoir matrix

Definition at line 626 of file fasp_block.h.

8.38.2.12 ivector* pivot_S

pivoting for the GS smoothers for saturation block

Definition at line 567 of file fasp_block.h.

8.38.2.13 dCSRmat* PP

pressure block

Definition at line 570 of file fasp_block.h.

8.38.2.14 dvector r

temporary dvector used to store and restore the residual

Definition at line 648 of file fasp_block.h.

8.38.2.15 INT restart

number of iterations for restart

Definition at line 644 of file fasp_block.h.

8.38.2.16 dBSRmat* RR

reservoir block

Definition at line 558 of file fasp_block.h.

8.38.2.17 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 556 of file fasp_block.h.

8.38.2.18 dBSRmat* SS

saturation block

Definition at line 565 of file fasp_block.h.

8.38.2.19 REAL tol

tolerance

Definition at line 645 of file fasp_block.h.

8.38.2.20 REAL* w

temporary work space for other usage

Definition at line 649 of file fasp_block.h.

8.38.2.21 dCSRmat* WW

Argumented well block

Definition at line 634 of file fasp_block.h.

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

- [fasp_block.h](#)

8.39 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](#)

8.39.1 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 662 of file fasp_block.h.

8.39.2 Field Documentation

8.39.2.1 block_dCSRmat* A

problem data, the sparse matrix

Definition at line 666 of file fasp_block.h.

8.39.2.2 block_dCSRmat* Ai

preconditioner data, the sparse matrix

Definition at line 667 of file fasp_block.h.

8.39.2.3 dCSRmat* local_A

local stiffness matrix for each layer

Definition at line 669 of file fasp_block.h.

8.39.2.4 ivector* local_index

local index for each layer

Definition at line 672 of file fasp_block.h.

8.39.2.5 void** local_LU

lcoal LU decomposition – (only for UMFpack – Xiaozhe Hu)

Definition at line 670 of file fasp_block.h.

8.39.2.6 INT NumLayers

number of layers

Definition at line 664 of file fasp_block.h.

8.39.2.7 dvector r

temporary dvector used to store and restore the residual

Definition at line 675 of file fasp_block.h.

8.39.2.8 REAL* w

temporary work space for other usage

Definition at line 676 of file fasp_block.h.

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

- [fasp_block.h](#)

8.40 Schwarz_data Struct Reference

Data for Schwarz methods.

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

Data Fields

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

8.40.1 Detailed Description

Data for Schwarz methods.

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

Definition at line 467 of file fasp.h.

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

- [fasp.h](#)

8.41 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

8.41.1 Detailed Description

Parameters for Schwarz method.

Added on 05/14/2012

Definition at line 424 of file fasp.h.

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

- [fasp.h](#)

Chapter 9

File Documentation

9.1 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.

9.1.1 Detailed Description

AMG method as an iterative solver (main file)

9.1.2 Function Documentation

9.1.2.1 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.

9.2 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.

9.2.1 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"

TODO: Not working. Yet need to be fixed. –Chensong

9.2.2 Function Documentation

9.2.2.1 [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.

9.3 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.

9.3.1 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.

9.3.2 Function Documentation

9.3.2.1 [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.

9.4 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)

9.4.1 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

9.4.2 Function Documentation

9.4.2.1 `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.

9.4.2.2 `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.

9.5 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)

9.5.1 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

9.5.2 Function Documentation

9.5.2.1 [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.

9.5.2.2 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.

9.6 amg_solve.c File Reference

Algebraic multigrid iterations: SOLVE phase.

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

Functions

- [INT fasp_amg_solve](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
AMG – SOLVE phase.
- [INT fasp_amg_solve_aml](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
AML – SOLVE phase.
- [INT fasp_amg_solve_nl_aml](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
Nonlinear AML – SOLVE phase.
- [void fasp_fm](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
FM – SOLVE phase.

9.6.1 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!

9.6.2 Function Documentation

9.6.2.1 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.

9.6.2.2 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.

9.6.2.3 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.

9.6.2.4 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.

9.7 amlirecur.c File Reference

Abstract AMLI multilevel iteration – recursive version.

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

9.7.1 Detailed Description

Abstract AMLI multilevel iteration – recursive version.

Note

AMLI and non-linear AMLI cycles

9.7.2 Function Documentation

9.7.2.1 void `fasp_amg_amli_coef` (const `REAL` *lambda_max*, const `REAL` *lambda_min*, const `INT` *degree*, `REAL` * *coef*)

Compute the coefficients of the polynomial used by AMLI-cycle.

Parameters

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

Author

Xiaozhe Hu

Date

01/23/2011

Definition at line 679 of file amlirecur.c.

9.7.2.2 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.

Definition at line 44 of file amlirecur.c.

9.7.2.3 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.

Definition at line 259 of file amlirecur.c.

9.7.2.4 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.

Definition at line 489 of file amlirecur.c.

9.8 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.

9.8.1 Detailed Description

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

9.8.2 Function Documentation

9.8.2.1 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 169 of file array.c.

9.8.2.2 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 209 of file array.c.

9.8.2.3 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 230 of file array.c.

9.8.2.4 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 253 of file array.c.

9.8.2.5 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.

9.8.2.6 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

Date

05/23/2012

Definition at line 49 of file array.c.

9.8.2.7 void fasp_iarray_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 189 of file array.c.

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

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

Parameters

n	Number of variables
x	Pointer to the vector
val	Initial value for the REAL array

Author

Chensong Zhang

Date

04/03/2010

Modified by Chunsheng Feng, Xiaoqiang Yue

Date

05/25/2012

Definition at line 111 of file array.c.

9.9 blas_array.c File Reference

BLAS 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.

9.9.1 Detailed Description

BLAS operations for arrays.

9.9.2 Function Documentation

9.9.2.1 void fasp_blas_array_ax (const INT *n*, const REAL *a*, REAL * *x*)

$x = a * x$

Parameters

<i>n</i>	Number of variables
<i>a</i>	Factor a
<i>x</i>	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.

9.9.2.2 void fasp_blas_array_axpby (const INT *n*, const REAL *a*, REAL * *x*, const REAL *b*, REAL * *y*)

$y = a * x + b * y$

Parameters

<i>n</i>	Number of variables
<i>a</i>	Factor a
<i>x</i>	Pointer to x
<i>b</i>	Factor b
<i>y</i>	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.

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

$y = a*x + y$

Parameters

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

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.

9.9.2.4 void fasp_blas_array_axpyz (const INT *n*, const REAL *a*, REAL * *x*, REAL * *y*, REAL * *z*)

$z = a*x + y$

Parameters

<i>n</i>	Number of variables
<i>a</i>	Factor a
<i>x</i>	Pointer to x
<i>y</i>	Pointer to y
<i>z</i>	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.

9.9.2.5 `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.

9.9.2.6 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.

9.9.2.7 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.

9.9.2.8 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.

9.10 blas_bcsr.c File Reference

BLAS operations for [block_dCSRmat](#) matrices.

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

Functions

- void `fasp_blas_bdcscr_aApy` (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_aApy` (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 $y = A * x$.*

9.10.1 Detailed Description

BLAS operations for `block_dCSRmat` matrices.

9.10.2 Function Documentation

9.10.2.1 void `fasp_blas_bdbsr_aApy` (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`.

9.10.2.2 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.

9.10.2.3 void fasp_blas_bdcscr_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 a
<i>A</i>	Pointer to block_dCSRmat matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

Author

Xiaozhe Hu

Date

06/04/2010

Definition at line 30 of file blas_bcsr.c.

9.10.2.4 void fasp_blas_bdcscr_mxv (block_dCSRmat * *A*, REAL * *x*, REAL * *y*)

Matrix-vector multiplication $y = A x$.

Parameters

<i>A</i>	Pointer to block_dCSRmat matrix A
<i>x</i>	Pointer to array x
<i>y</i>	Pointer to array y

Author

Chensong Zhang

Date

04/27/2013

Definition at line 155 of file blas_bcsr.c.

9.11 blas_bsr.c File Reference

BLAS operations for [dBSRmat](#) matrices.

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

Functions

- void `fasp_blas_dbsr_axm` (`dBSRmat *A`, const `REAL alpha`)
Multiply a sparse matrix A in BSR format by a scalar alpha.
- void `fasp_blas_dbsr_aAxpby` (const `REAL alpha`, `dBSRmat *A`, `REAL *x`, const `REAL beta`, `REAL *y`)
*Compute $y := \alpha * A * x + \beta * y$.*
- void `fasp_blas_dbsr_aAxy` (const `REAL alpha`, `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!*

9.11.1 Detailed Description

BLAS operations for `dBSRmat` matrices.

9.11.2 Function Documentation

9.11.2.1 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

Date

06/29/2012

Note

Works for general nb (Xiaozhe)

Definition at line 60 of file blas_bsr.c.

9.11.2.2 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

Date

05/23/2012

Note

Works for general nb (Xiaozhe)

Definition at line 339 of file blas_bsr.c.

9.11.2.3 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 612 of file blas_bsr.c.

9.11.2.4 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.

9.11.2.5 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 4595 of file blas_bsr.c.

9.11.2.6 void fasp_blas_dbsr_mnv (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 897 of file blas_bsr.c.

9.11.2.7 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 2643 of file blas_bsr.c.

9.11.2.8 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 4899 of file blas_bsr.c.

9.11.2.9 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 4715 of file blas_bsr.c.

9.11.2.10 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

R	Pointer to the dBSRmat matrix
A	Pointer to the dBSRmat matrix
P	Pointer to the dBSRmat matrix
B	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 5164 of file blas_bsr.c.

9.12 blas_csr.c File Reference

BLAS 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_mxv](#) ([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.

9.12.1 Detailed Description

BLAS operations for `dCSRmat` matrices.

Note

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

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

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

9.12.2 Function Documentation

9.12.2.1 void `fasp_blas_dcsr_aAxy` (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 <code>dCSRmat</code> 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 480 of file blas_csr.c.

9.12.2.2 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 594 of file blas_csr.c.

9.12.2.3 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.

9.12.2.4 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 A
<i>alpha</i>	REAL factor alpha

Author

Chensong Zhang

Date

07/01/2009

Modified by Chunsheng Feng, Zheng Li on 06/29/2012

Definition at line 201 of file blas_csr.c.

9.12.2.5 fasp_blas_dcsr_bandwith ([dCSRmat](#) * *A*, INT * *bndwith*)

Get bandwith of matrix.

Parameters

<i>A</i>	pointer to the dCSRmat matrix
<i>bndwith</i>	pointer to the bandwith

Author

Zheng Li

Date

03/22/2015

Definition at line 2000 of file blas_csr.c.

9.12.2.6 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 760 of file blas_csr.c.

9.12.2.7 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.

9.12.2.8 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

<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 423 of file blas_csr.c.

9.12.2.9 void fasp_blas_dcsr_ptap (dCSRmat * Pt, dCSRmat * A, dCSRmat * P, dCSRmat * Ac)

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

Parameters

<i>Pt</i>	Pointer to the restriction matrix
-----------	-----------------------------------

<i>A</i>	Pointer to the fine coefficient matrix
<i>P</i>	Pointer to the prolongation matrix
<i>Ac</i>	Pointer to the coarse coefficient matrix (output)

Author

Ludmil Zikatanov, Chensong Zhang

Date

05/10/2010

Modified by Chunsheng Feng, Zheng Li on 10/19/2012

Note

Driver to compute triple matrix product P^*A^*P using Itz CSR format. In Itz format: $ia[0]=1$, $ja[0]$ and $a[0]$ are used as usual. When called from Fortran, $ia[0]$, $ja[0]$ and $a[0]$ will be just $ia(1), ja(1), a(1)$. For the indices, $ia_itz[k] = ia_usual[k]+1$, $ja_itz[k] = ja_usual[k]+1$, $a_itz[k] = a_usual[k]$.

Definition at line 1597 of file blas_csr.c.

9.12.2.10 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 867 of file blas_csr.c.

9.12.2.11 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 1699 of file blas_csr.c.

9.12.2.12 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 1149 of file blas_csr.c.

9.12.2.13 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 1414 of file blas_csr.c.

9.12.2.14 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 705 of file blas_csr.c.

9.13 blas_csrl.c File Reference

BLAS operations for [dCSRLmat](#) matrices.

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

Functions

- void [fasp_blas_dcsrl_mxv](#) ([dCSRLmat](#) *A, [REAL](#) *x, [REAL](#) *y)
Compute $y = A*x$ for a sparse matrix in CSRL format.

9.13.1 Detailed Description

BLAS operations for [dCSRmat](#) matrices.

Note

For details of CSR 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.

9.13.2 Function Documentation

9.13.2.1 void fasp_blas_dcsr_mv ([dCSRmat](#) * A, REAL * x, REAL * y)

Compute $y = A \cdot x$ for a sparse matrix in CSR format.

Parameters

A	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_csr.c.

9.14 blas_smat.c File Reference

BLAS operations for *small* dense matrix.

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

Functions

- void [fasp_blas_smat_axm](#) (REAL *a, const INT n, const REAL alpha)
Compute $\alpha \cdot a$, store in a.
- void [fasp_blas_smat_add](#) (REAL *a, REAL *b, const INT n, const REAL alpha, const REAL beta, REAL *c)
Compute $c = \alpha \cdot a + \beta \cdot b$.
- void [fasp_blas_smat_mv_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_mv_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_mv_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_mv_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_mv](#) (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_mul_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_mul_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_mul_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_mul_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_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.*

 - void `fasp_blas_array_axpyz_nc2` (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, 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 $ys := ys - Ass * xs$, 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 $ys := ys - Ass * xs$, 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 $ys := ys - Ass * xs$, 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 $ys := ys - Ass * xs$, 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 $ys := ys - Ass * xs$, where 'A' is a n*n dense matrix, Ass is its saturaton part (n-1)*(n-1).

9.14.1 Detailed Description

BLAS operations for *small* dense matrix.

9.14.2 Function Documentation

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

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

Parameters

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

Author

Xiaozhe Hu

Date

18/11/2011

Definition at line 683 of file blas_smat.c.

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

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

Parameters

<i>a</i>	REAL factor a
----------	---------------

x	Pointer to the original array
y	Pointer to the destination array

Author

Xiaozhe Hu, Shiquan Zhang

Date

05/01/2010

Definition at line 706 of file blas_smat.c.

9.14.2.3 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

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 735 of file blas_smat.c.

9.14.2.4 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

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 782 of file blas_smat.c.

9.14.2.5 void fasp_blas_array_axpyz_nc2 (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 498 of file blas_smat.c.

9.14.2.6 void fasp_blas_array_axpyz_nc3 (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 3

Definition at line 525 of file blas_smat.c.

9.14.2.7 void fasp_blas_array_axpyz_nc5 (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 5

Definition at line 558 of file blas_smat.c.

9.14.2.8 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 609 of file blas_smat.c.

9.14.2.9 void fasp_blas_smat_aAxpby (const REAL *alpha*, REAL * *A*, REAL * *x*, const REAL *beta*, REAL * *y*, const INT *n*)

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

Parameters

<i>alpha</i>	REAL factor alpha
<i>A</i>	Pointer to the REAL array which stands for a n*n full matrix
<i>x</i>	Pointer to the REAL array with length n
<i>beta</i>	REAL factor beta
<i>y</i>	Pointer to the REAL array with length n
<i>n</i>	Length of array x and y

Author

Zhiyang Zhou

Date

2010/10/25

Definition at line 1306 of file blas_smat.c.

9.14.2.10 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 52 of file blas_smat.c.

9.14.2.11 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
----------	---

n	Dimension of the matrix
α	Scalar

Author

Xiaozhe Hu

Date

05/26/2014

Definition at line 24 of file blas_smat.c.

9.14.2.12 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 $n \times n$ matrix
<i>b</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>c</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>n</i>	Dimension of the matrix

Author

Xiaozhe Hu, Shiquan Zhang

Date

04/21/2010

Definition at line 446 of file blas_smat.c.

9.14.2.13 void fasp_blas_smat_mul_nc2 (REAL * *a*, REAL * *b*, REAL * *c*)

Compute the matrix product of two $2 \times$ matrices *a* and *b*, stored in *c*.

Parameters

<i>a</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>b</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>c</i>	Pointer to the REAL array which stands a $n \times n$ matrix

Author

Xiaozhe Hu

Date

18/11/2011

Definition at line 231 of file blas_smat.c.

9.14.2.14 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 $n \times n$ matrix
<i>b</i>	Pointer to the REAL array which stands a $n \times n$ matrix
<i>c</i>	Pointer to the REAL array which stands a $n \times n$ matrix

Author

Xiaozhe Hu, Shiquan Zhang

Date

05/01/2010

Definition at line 260 of file blas_smat.c.

9.14.2.15 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 297 of file blas_smat.c.

9.14.2.16 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 356 of file blas_smat.c.

9.14.2.17 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 $n \times n$ 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 181 of file blas_smat.c.

9.14.2.18 void fasp_blas_smat_mnv_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 81 of file blas_smat.c.

9.14.2.19 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 103 of file blas_smat.c.

9.14.2.20 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 126 of file blas_smat.c.

9.14.2.21 void fasp_blas_smat_mnv_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 152 of file blas_smat.c.

9.14.2.22 void fasp_blas_smat_ymAx (REAL * *A*, REAL * *x*, REAL * *y*, INT *n*)

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

Parameters

<i>A</i>	Pointer to the <i>n</i> * <i>n</i> 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 1205 of file blas_smat.c.

9.14.2.23 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

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

Author

Xiaozhe Hu

Date

18/11/2011

Note

Works for 2-component

Definition at line 1075 of file blas_smat.c.

9.14.2.24 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 1103 of file blas_smat.c.

9.14.2.25 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 1133 of file blas_smat.c.

9.14.2.26 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 1167 of file blas_smat.c.

9.14.2.27 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 $(n-1) * (n-1)$.

Parameters

A	Pointer to the $n \times n$ 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 1480 of file blas_smat.c.

9.14.2.28 void fasp_blas_smat_ymAx_ns2 (REAL * A , REAL * x , REAL * y)

Compute $ys := ys - Ass \cdot xs$, where ' A ' is a 2×2 dense matrix, Ass is its saturation 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 1356 of file blas_smat.c.

9.14.2.29 void fasp_blas_smat_ymAx_ns3 (REAL * A , REAL * x , REAL * y)

Compute $ys := ys - Ass \cdot xs$, where ' A ' is a 3×3 dense matrix, Ass is its saturation part 2×2 .

Parameters

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

Author

Xiaozhe Hu

Date

2010/10/25

Note

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

Definition at line 1380 of file blas_smat.c.

9.14.2.30 void fasp_blas_smat_ymAx_ns5 (REAL * A , REAL * x , REAL * y)

Compute $ys := ys - Ass * xs$, 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 1408 of file blas_smat.c.

9.14.2.31 void fasp_blas_smat_ymAx_ns7 (REAL * A , REAL * x , REAL * y)

Compute $ys := ys - Ass * xs$, where ' A ' is a 7*7 dense matrix, Ass is its saturaton 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 1442 of file blas_smat.c.

9.14.2.32 void fasp_blas_smat_ypAx (REAL * A , REAL * x , REAL * y , const INT n)

Compute $y := y + Ax$, where ' A ' is a $n \times n$ 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 974 of file blas_smat.c.

9.14.2.33 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 855 of file blas_smat.c.

9.14.2.34 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 881 of file blas_smat.c.

9.14.2.35 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**

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

Author

Zhiyang Zhou, Xiaozhe Hu

Date

2010/10/25

Definition at line 908 of file blas_smat.c.

9.14.2.36 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 939 of file blas_smat.c.

9.15 blas_str.c File Reference

BLAS operations for [dSTRmat](#) matrices.

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

Functions

- void [fasp_blas_dstr_aAxy](#) (REAL alpha, [dSTRmat](#) *A, REAL *x, REAL *y)
*Matrix-vector multiplication $y = \alpha * A * x + y$.*
- void [fasp_blas_dstr_mxy](#) ([dSTRmat](#) *A, REAL *x, REAL *y)
*Matrix-vector multiplication $y = A * x$.*
- INT [fasp_dstr_diagscale](#) ([dSTRmat](#) *A, [dSTRmat](#) *B)
 $B = D^{-1} A$.

9.15.1 Detailed Description

BLAS operations for [dSTRmat](#) matrices.

9.15.2 Function Documentation

9.15.2.1 void [fasp_blas_dstr_aAxy](#) (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.

9.15.2.2 void fasp_blas_dstr_mxv (dSTRmat * A, REAL * x, REAL * y)Matrix-vector multiplication $y = A*x$.**Parameters**

<i>A</i>	Pointer to 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.

9.15.2.3 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.

9.16 blas_vec.c File Reference

BLAS 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.

9.16.1 Detailed Description

BLAS operations for vectors.

9.16.2 Function Documentation

9.16.2.1 void [fasp_blas_dvec_axpy](#) (const [REAL](#) a, [dvector](#) * x, [dvector](#) * y)

$y = a*x + y$

Parameters

<i>a</i>	REAL factor a
----------	---------------

x	Pointer to dvector x
y	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.

9.16.2.2 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**

a	REAL factor a
x	Pointer to dvector x
y	Pointer to dvector y
z	Pointer to dvector z

Author

Chensong Zhang

Date

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue

Date

05/23/2012

Definition at line 85 of file blas_vec.c.

9.16.2.3 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.

9.16.2.4 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.

9.16.2.5 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.

9.16.2.6 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.

9.16.2.7 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

07/01/2009

Modified by Chunsheng Feng, Xiaoqiang Yue

Date

05/23/2012

Definition at line 167 of file blas_vec.c.

9.17 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 whether 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.

9.17.1 Detailed Description

Check matrix properties.

9.17.2 Function Documentation

9.17.2.1 SHORT fasp_check_dCSRmat (dCSRmat * A)

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

Parameters

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

Author

Shuo Zhang

Date

03/29/2009

Definition at line 275 of file checkmat.c.

9.17.2.2 INT fasp_check_diagdom (dCSRmat * A)

Check whether a matrix is diagonal dominant.

INT fasp_check_diagdom ([dCSRmat](#) *A)

Parameters

<i>A</i>	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.

9.17.2.3 INT fasp_check_diagpos (dCSRmat * A)

Check positivity of diagonal entries of a CSR sparse matrix.

Parameters

A	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.

9.17.2.4 SHORT fasp_check_diagzero (dCSRmat * A)

Check whether a CSR sparse matrix has diagonal entries that are very close to zero.

Parameters

A	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.

9.17.2.5 SHORT fasp_check_iCSRmat (iCSRmat * A)

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

Parameters

A	Pointer to the matrix in iCSRmat format
-----	---

Author

Shuo Zhang

Date

03/29/2009

Definition at line 309 of file checkmat.c.

9.17.2.6 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.

9.18 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](#) (INT *i_0*, INT *i_n*, [dCSRmat](#) **A*, [ivector](#) **vertices*, [AMG_param](#) **param*)
CR coarsening.

9.18.1 Detailed Description

Coarsening with Brannick-Falgout strategy.

9.18.2 Function Documentation

9.18.2.1 [INT fasp_amg_coarsening_cr](#) (INT *i_0*, 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.

9.19 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.

9.19.1 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!!!

9.19.2 Function Documentation

9.19.2.1 **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.

9.20 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, [INT](#) vlength, [INT](#) endianflag)
Swap order of a REAL number.

9.20.1 Detailed Description

Some utilities for format conversion.

9.20.2 Function Documentation

9.20.2.1 `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`.

9.20.2.2 `REAL endian_convert_real (const REAL rnum, INT ilength, 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`.

9.20.2.3 `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.

9.20.2.4 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.

9.20.2.5 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.

9.21 doxygen.h File Reference

Main page for Doygen documentation.

9.21.1 Detailed Description

Main page for Doygen documentation.

9.22 eigen.c File Reference

Simple subroutines for compute 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.

9.22.1 Detailed Description

Simple subroutines for compute the extreme eigenvalues.

9.22.2 Function Documentation

9.22.2.1 [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.

9.23 factor.f File Reference

LU factorization for CSR matrix.

Functions/Subroutines

- subroutine **sfactr** (ia, ja, n, iu, ju, ip, nwku)
- subroutine **sfactr_new** (ia, ja, n, iu, ju, ip, nwku, mem_chk)
- subroutine **factor** (ia, ja, n, iu, ju, ip, iup, an, ad, un, di)
- subroutine **forbac** (iu, ju, un, di, n, x)

9.23.1 Detailed Description

LU factorization for CSR matrix.

Author

Ludmil Zikatanov

Date

01/01/2002

9.24 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.

9.24.1 Detailed Description

full AMG method as an iterative solver (main file)

9.24.2 Function Documentation

9.24.2.1 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.

9.25 fasp.h File Reference

Main header file for FASP.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "fasp_const.h"
```

Data Structures

- struct [ddenmat](#)
Dense matrix of REAL type.
- struct [idenmat](#)
Dense matrix of INT type.

- struct [dCSRmat](#)
Sparse matrix of REAL type in CSR format.
- struct [iCSRmat](#)
Sparse matrix of INT type in CSR format.
- struct [dCOOmat](#)
Sparse matrix of REAL type in COO (or IJ) format.
- struct [iCOOmat](#)
Sparse matrix of INT type in COO (or IJ) format.
- struct [dCSRLmat](#)
Sparse matrix of REAL type in CSRL format.
- struct [dSTRmat](#)
Structure matrix of REAL type.
- struct [dvector](#)
Vector with n entries of REAL type.
- struct [ivector](#)
Vector with n entries of INT type.
- struct [ILU_param](#)
Parameters for ILU.
- struct [ILU_data](#)
Data for ILU setup.
- struct [Schwarz_param](#)
Parameters for Schwarz method.
- struct [Mumps_data](#)
Parameters for MUMPS interface.
- struct [Schwarz_data](#)
Data for Schwarz methods.
- struct [AMG_param](#)
Parameters for AMG solver.
- struct [AMG_data](#)
Data for AMG solvers.
- struct [precond_data](#)
Data passed to the preconditioners.
- struct [precond_data_str](#)
Data passed to the preconditioner for [dSTRmat](#) matrices.
- struct [precond_diagstr](#)
Data passed to diagonal preconditioner for [dSTRmat](#) matrices.
- struct [precond](#)
Preconditioner data and action.
- struct [mxv_matfree](#)
Matrix-vector multiplication, replace the actual matrix.
- struct [input_param](#)
Input parameters.
- struct [itsolver_param](#)
Parameters passed to iterative solvers.
- struct [grid2d](#)
Two dimensional grid data structure.
- struct [Link](#)
Struct for Links.
- struct [linked_list](#)
A linked list node.

Macros

- #define `__FASP_HEADER__`
- #define `FASP_USE_ILU ON`
 - For external software package support.*
- #define `DLMALLOC OFF`
- #define `NEDMALLOC OFF`
- #define `RS_C1 ON`
 - Flags for internal uses (change with caution!!!)*
- #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`

9.25.1 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 decorations.

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

9.25.2 Macro Definition Documentation

9.25.2.1 `#define __FASP_HEADER__`

indicate `fasp.h` has been included before

Definition at line 28 of file `fasp.h`.

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

absolute value of a

Definition at line 64 of file `fasp.h`.

9.25.2.3 `#define DIAGONAL_PREF OFF`

order each row such that diagonal appears first

Definition at line 48 of file `fasp.h`.

9.25.2.4 `#define DLMALLOC OFF`

use `dmalloc` instead of standard `malloc`

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

9.25.2.5 `#define FASP_GSRB 1`

MG level 0 use RedBlack Gauss Seidel Smoothing

Definition at line 1158 of file fasp.h.

9.25.2.6 `#define FASP_USE_ILU ON`

For external software package support.

enable ILU or not

Definition at line 37 of file fasp.h.

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

is $a \geq b$?

Definition at line 70 of file fasp.h.

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

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

is $a > b$?

Definition at line 69 of file fasp.h.

9.25.2.9 `#define INT int`

regular integer type: int or long

Definition at line 54 of file fasp.h.

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

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

Definition at line 73 of file fasp.h.

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

is $a \leq b$?

Definition at line 72 of file fasp.h.

9.25.2.12 `#define LONG long`

long integer type

Definition at line 55 of file fasp.h.

9.25.2.13 #define LONGLONG long long

long integer type

Definition at line 56 of file fasp.h.

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

is a < b?

Definition at line 71 of file fasp.h.

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

Definition of max, min, abs.

bigger one in a and b

Definition at line 62 of file fasp.h.

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

smaller one in a and b

Definition at line 63 of file fasp.h.

9.25.2.17 #define NEDMALLOC OFF

use nedmalloc instead of standard malloc

Definition at line 39 of file fasp.h.

9.25.2.18 #define PUT_INT(A) printf("### DEBUG: %s = %d\n", #A, (A))

Definition of print command in DEBUG mode.

print an integer

Definition at line 78 of file fasp.h.

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

print a real num

Definition at line 79 of file fasp.h.

9.25.2.20 #define REAL double

float type

Definition at line 57 of file fasp.h.

9.25.2.21 `#define RS_C1 ON`

Flags for internal uses (change with caution!!!)

CF splitting of RS: check C1 Criterion

Definition at line 46 of file fasp.h.

9.25.2.22 `#define SHORT short`

FASP integer and floating point numbers.

short integer type

Definition at line 53 of file fasp.h.

9.25.3 Typedef Documentation

9.25.3.1 `typedef struct dCOOmat dCOOmat`

Sparse matrix of REAL type in COO format

9.25.3.2 `typedef struct dCSRLmat dCSRLmat`

Sparse matrix of REAL type in CSRL format

9.25.3.3 `typedef struct dCSRmat dCSRmat`

Sparse matrix of REAL type in CSR format

9.25.3.4 `typedef struct ddenmat ddenmat`

Dense matrix of REAL type

9.25.3.5 `typedef struct dSTRmat dSTRmat`

Structured matrix of REAL type

9.25.3.6 `typedef struct dvector dvector`

Vector of REAL type

9.25.3.7 `typedef struct grid2d grid2d`

2D grid type for plotting

9.25.3.8 `typedef struct iCOOmat iCOOmat`

Sparse matrix of INT type in COO format

9.25.3.9 `typedef struct iCSRmat iCSRmat`

Sparse matrix of INT type in CSR format

9.25.3.10 `typedef struct idenmat idenmat`

Dense matrix of INT type

9.25.3.11 `typedef struct ivector ivector`

Vector of INT type

9.25.3.12 `typedef ListElement* LinkList`

List of linkslinked list

Definition at line 1153 of file fasp.h.

9.25.3.13 `typedef struct linked_list ListElement`

Linked element in list

9.25.3.14 `typedef const grid2d* pcgrid2d`

Grid in 2d

Definition at line 1107 of file fasp.h.

9.25.3.15 `typedef grid2d* pgrid2d`

Grid in 2d

Definition at line 1105 of file fasp.h.

9.25.4 Variable Documentation

9.25.4.1 `INT count`

Counter for multiple calls

9.25.4.2 `INT* IMAP`

Red Black Gs Smoother imap

9.25.4.3 `INT MAXIMAP`

Red Black Gs Smoother max DOFs of reservoir

9.25.4.4 INT nx_rb

Red Black Gs Smoother Nx

9.25.4.5 INT ny_rb

Red Black Gs Smoother Ny

9.25.4.6 INT nz_rb

Red Black Gs Smoother Nz

9.25.4.7 unsigned INT total_alloc_count

Total allocated memory amount.

total allocation times

Definition at line 33 of file memory.c.

9.25.4.8 unsigned INT total_alloc_mem

total allocated memory

Definition at line 32 of file memory.c.

9.26 fasp_block.h File Reference

Main 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 preconditioning 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.

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_ivector](#) [block_ivector](#)
- typedef struct [block_Reservoir](#) [block_Reservoir](#)
- typedef struct [block_BSR](#) [block_BSR](#)
- typedef struct [precond_block_reservoir_data](#) [precond_block_reservoir_data](#)

9.26.1 Detailed Description

Main 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 decorations.

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.

9.26.2 Typedef Documentation

9.26.2.1 typedef struct [block_BSR](#) [block_BSR](#)

Block of BSR matrices of REAL type

9.26.2.2 typedef struct [block_dCSRmat](#) [block_dCSRmat](#)

Matrix of REAL type in Block CSR format

9.26.2.3 typedef struct **block_dvector** **block_dvector**

Vector of REAL type in Block format

9.26.2.4 typedef struct **block_iCSRmat** **block_iCSRmat**

Matrix of INT type in Block CSR format

9.26.2.5 typedef struct **block_ivector** **block_ivector**

Vector of INT type in Block format

9.26.2.6 typedef struct **block_Reservoir** **block_Reservoir**

Special block matrix for Reservoir Simulation

9.26.2.7 typedef struct **dBSRmat** **dBSRmat**

Matrix of REAL type in BSR format

9.26.2.8 typedef struct **precond_block_reservoir_data** **precond_block_reservoir_data**

Precond data for Reservoir Simulation

9.27 fasp_const.h File Reference

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

Macros

- #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 **STAG_RATIO** 1e-4
- #define **MAX_STAG** 20
- #define **MAX_RESTART** 20
- #define **OPENMP_HOLDS** 2000
- #define **FASP_SUCCESS** 0
Definition of return status and error messages.
- #define **ERROR_OPEN_FILE** -10

- #define [ERROR_WRONG_FILE](#) -11
 - #define [ERROR_INPUT_PAR](#) -13
 - #define [ERROR_REGRESS](#) -14
 - #define [ERROR_MAT_SIZE](#) -15
 - #define [ERROR_NUM_BLOCKS](#) -18
 - #define [ERROR_MISC](#) -19
 - #define [ERROR_ALLOC_MEM](#) -20
 - #define [ERROR_DATA_STRUCTURE](#) -21
 - #define [ERROR_DATA_ZERODIAG](#) -22
 - #define [ERROR_DUMMY_VAR](#) -23
 - #define [ERROR_AMG_INTERP_TYPE](#) -30
 - #define [ERROR_AMG_SMOOTH_TYPE](#) -31
 - #define [ERROR_AMG_COARSE_TYPE](#) -32
 - #define [ERROR_AMG_COARSEING](#) -33
 - #define [ERROR_SOLVER_TYPE](#) -40
 - #define [ERROR_SOLVER_PRECTYPE](#) -41
 - #define [ERROR_SOLVER_STAG](#) -42
 - #define [ERROR_SOLVER_SOLSTAG](#) -43
 - #define [ERROR_SOLVER_TOLSMALL](#) -44
 - #define [ERROR_SOLVER_ILUSETUP](#) -45
 - #define [ERROR_SOLVER_MISC](#) -46
 - #define [ERROR_SOLVER_MAXIT](#) -48
 - #define [ERROR_SOLVER_EXIT](#) -49
 - #define [ERROR_QUAD_TYPE](#) -60
 - #define [ERROR_QUAD_DIM](#) -61
 - #define [ERROR_LIC_TYPE](#) -80
 - #define [ERROR_UNKNOWN](#) -99
 - #define [TRUE](#) 1
- Definition of logic type.*
- #define [FALSE](#) 0
 - #define [ON](#) 1
- Definition of switch.*
- #define [OFF](#) 0
 - #define [PRINT_NONE](#) 0
- Print level for all subroutines – not including DEBUG output.*
- #define [PRINT_MIN](#) 1
 - #define [PRINT_SOME](#) 2
 - #define [PRINT_MORE](#) 4
 - #define [PRINT_MOST](#) 8
 - #define [PRINT_ALL](#) 10
 - #define [MAT_FREE](#) 0
- Definition of matrix format.*
- #define [MAT_CSR](#) 1
 - #define [MAT_BSR](#) 2
 - #define [MAT_STR](#) 3
 - #define [MAT_bCSR](#) 4
 - #define [MAT_bBSR](#) 5
 - #define [MAT_CSRL](#) 6
 - #define [MAT_SymCSR](#) 7
 - #define [SOLVER_DEFAULT](#) 0

Definition of solver types for iterative methods.

- #define SOLVER_CG 1
- #define SOLVER_BiCGstab 2
- #define SOLVER_MinRes 3
- #define SOLVER_GMRES 4
- #define SOLVER_VGMRES 5
- #define SOLVER_VFGMRES 6
- #define SOLVER_GCG 7
- #define SOLVER_GCR 8
- #define SOLVER_SCG 11
- #define SOLVER_SBiCGstab 12
- #define SOLVER_SMinRes 13
- #define SOLVER_SGMRES 14
- #define SOLVER_SVGMRES 15
- #define SOLVER_SVFGMRES 16
- #define SOLVER_SGCG 17
- #define SOLVER_AMG 21
- #define SOLVER_FMG 22
- #define SOLVER_SUPERLU 31
- #define SOLVER_UMFPACK 32
- #define SOLVER_MUMPS 33
- #define STOP_REL_RES 1

Definition of iterative solver stopping criteria types.

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

Definition of preconditioner type for iterative methods.

- #define PREC_DIAG 1
- #define PREC_AMG 2
- #define PREC_FMG 3
- #define PREC_ILU 4
- #define PREC_SCHWARZ 5
- #define ILUk 1

Type of ILU methods.

- #define ILU_t 2
- #define ILU_{tp} 3
- #define SCHWARZ_FORWARD 1

Type of Schwarz smoother.

- #define SCHWARZ_BACKWARD 2
- #define SCHWARZ_SYMMETRIC 3
- #define CLASSIC_AMG 1

Definition of AMG types.

- #define SA_AMG 2
- #define UA_AMG 3
- #define PAIRWISE 1

Definition of aggregation types.

- #define VMB 2
- #define V_CYCLE 1

Definition of cycle types.

- #define W_CYCLE 2

- #define [AMLI_CYCLE](#) 3
- #define [NL_AMLI_CYCLE](#) 4
- #define [SMOOTHER_JACOBI](#) 1

Definition of standard smoother types.

- #define [SMOOTHER_GS](#) 2
- #define [SMOOTHER_SGS](#) 3
- #define [SMOOTHER_CG](#) 4
- #define [SMOOTHER_SOR](#) 5
- #define [SMOOTHER_SSOR](#) 6
- #define [SMOOTHER_GSOR](#) 7
- #define [SMOOTHER_SGSOR](#) 8
- #define [SMOOTHER_POLY](#) 9
- #define [SMOOTHER_L1DIAG](#) 10
- #define [SMOOTHER_BLKOil](#) 11

Definition of specialized smoother types.

- #define [SMOOTHER_SPETEN](#) 19
- #define [COARSE_RS](#) 1

Definition of coarsening types.

- #define [COARSE_RSP](#) 2
- #define [COARSE_CR](#) 3
- #define [COARSE_AC](#) 4
- #define [COARSE_MIS](#) 5
- #define [INTERP_DIR](#) 1

Definition of interpolation types.

- #define [INTERP_STD](#) 2
- #define [INTERP_ENG](#) 3
- #define [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

9.27.1 Detailed Description

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

Note

This is internal use only. Do NOT change.

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

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

9.27.2 Macro Definition Documentation

9.27.2.1 `#define AMLI_CYCLE 3`

AMLI-cycle

Definition at line 192 of file fasp_const.h.

9.27.2.2 `#define ASCEND 12`

Ascending order

Definition at line 252 of file fasp_const.h.

9.27.2.3 `#define BIGREAL 1e+20`

Some global constants.

A large real number

Definition at line 27 of file fasp_const.h.

9.27.2.4 `#define CF_ORDER 1`

C/F order smoothing

Definition at line 244 of file fasp_const.h.

9.27.2.5 `#define CGPT 1`

Coarse grid points

Definition at line 237 of file fasp_const.h.

9.27.2.6 `#define CLASSIC_AMG 1`

Definition of AMG types.

classic AMG

Definition at line 177 of file fasp_const.h.

9.27.2.7 #define COARSE_AC 4

Aggressive coarsening

Definition at line 221 of file fasp_const.h.

9.27.2.8 #define COARSE_CR 3

Compatible relaxation

Definition at line 220 of file fasp_const.h.

9.27.2.9 #define COARSE_MIS 5

Aggressive coarsening based on MIS

Definition at line 222 of file fasp_const.h.

9.27.2.10 #define COARSE_RS 1

Definition of coarsening types.

Classical coarsening

Definition at line 218 of file fasp_const.h.

9.27.2.11 #define COARSE_RSP 2

Classical coarsening with positive offdiags

Definition at line 219 of file fasp_const.h.

9.27.2.12 #define CPFIRST 1

C-points first order

Definition at line 250 of file fasp_const.h.

9.27.2.13 #define DESCEND 21

Descending order

Definition at line 253 of file fasp_const.h.

9.27.2.14 #define ERROR_ALLOC_MEM -20

fail to allocate memory

Definition at line 53 of file fasp_const.h.

9.27.2.15 #define ERROR_AMG_COARSE_TYPE -32

unknown coarsening type

Definition at line 60 of file fasp_const.h.

9.27.2.16 #define ERROR_AMG_COARSEING -33

coarsening step failed to complete

Definition at line 61 of file fasp_const.h.

9.27.2.17 #define ERROR_AMG_INTERP_TYPE -30

unknown interpolation type

Definition at line 58 of file fasp_const.h.

9.27.2.18 #define ERROR_AMG_SMOOTH_TYPE -31

unknown smoother type

Definition at line 59 of file fasp_const.h.

9.27.2.19 #define ERROR_DATA_STRUCTURE -21

problem with data structures

Definition at line 54 of file fasp_const.h.

9.27.2.20 #define ERROR_DATA_ZERODIAG -22

matrix has zero diagonal entries

Definition at line 55 of file fasp_const.h.

9.27.2.21 #define ERROR_DUMMY_VAR -23

unexpected input data

Definition at line 56 of file fasp_const.h.

9.27.2.22 #define ERROR_INPUT_PAR -13

wrong input argument

Definition at line 47 of file fasp_const.h.

9.27.2.23 #define ERROR_LIC_TYPE -80

wrong license type

Definition at line 76 of file fasp_const.h.

9.27.2.24 #define ERROR_MAT_SIZE -15

wrong problem size

Definition at line 49 of file fasp_const.h.

9.27.2.25 #define ERROR_MISC -19

other error

Definition at line 51 of file fasp_const.h.

9.27.2.26 #define ERROR_NUM_BLOCKS -18

wrong number of blocks

Definition at line 50 of file fasp_const.h.

9.27.2.27 #define ERROR_OPEN_FILE -10

fail to open a file

Definition at line 45 of file fasp_const.h.

9.27.2.28 #define ERROR_QUAD_DIM -61

unsupported quadrature dim

Definition at line 74 of file fasp_const.h.

9.27.2.29 #define ERROR_QUAD_TYPE -60

unknown quadrature type

Definition at line 73 of file fasp_const.h.

9.27.2.30 #define ERROR_REGRESS -14

regression test fail

Definition at line 48 of file fasp_const.h.

9.27.2.31 #define ERROR_SOLVER_EXIT -49

solver does not quit successfully

Definition at line 71 of file fasp_const.h.

9.27.2.32 #define ERROR_SOLVER_ILUSETUP -45

ILU setup error

Definition at line 68 of file fasp_const.h.

9.27.2.33 #define ERROR_SOLVER_MAXIT -48

maximal iteration number exceeded

Definition at line 70 of file fasp_const.h.

9.27.2.34 #define ERROR_SOLVER_MISC -46

misc solver error during run time

Definition at line 69 of file fasp_const.h.

9.27.2.35 #define ERROR_SOLVER_PRECTYPE -41

unknown precondition type

Definition at line 64 of file fasp_const.h.

9.27.2.36 #define ERROR_SOLVER_SOLSTAG -43

solver's solution is too small

Definition at line 66 of file fasp_const.h.

9.27.2.37 #define ERROR_SOLVER_STAG -42

solver stagnates

Definition at line 65 of file fasp_const.h.

9.27.2.38 #define ERROR_SOLVER_TOLSMALL -44

solver's tolerance is too small

Definition at line 67 of file fasp_const.h.

9.27.2.39 #define ERROR_SOLVER_TYPE -40

unknown solver type

Definition at line 63 of file fasp_const.h.

9.27.2.40 #define ERROR_UNKNOWN -99

an unknown error type

Definition at line 78 of file fasp_const.h.

9.27.2.41 #define ERROR_WRONG_FILE -11

input contains wrong format

Definition at line 46 of file fasp_const.h.

9.27.2.42 #define FALSE 0

logic FALSE

Definition at line 84 of file fasp_const.h.

9.27.2.43 #define FASP_SUCCESS 0

Definition of return status and error messages.

return from function successfully

Definition at line 43 of file fasp_const.h.

9.27.2.44 #define FGPT 0

Fine grid points

Definition at line 236 of file fasp_const.h.

9.27.2.45 #define FPFIRST -1

F-points first order

Definition at line 251 of file fasp_const.h.

9.27.2.46 #define G0PT -5

Type of vertices (DOFs) for coarsening.

Cannot fit in aggregates

Definition at line 234 of file fasp_const.h.

9.27.2.47 #define ILUk 1

Type of ILU methods.

ILUk

Definition at line 163 of file fasp_const.h.

9.27.2.48 #define ILUt 2

ILUt

Definition at line 164 of file fasp_const.h.

9.27.2.49 #define ILUtp 3

ILUtp

Definition at line 165 of file fasp_const.h.

9.27.2.50 #define INTERP_DIR 1

Definition of interpolation types.

Direct interpolation

Definition at line 227 of file fasp_const.h.

9.27.2.51 #define INTERP_ENG 3

energy minimization interpolation

Definition at line 229 of file fasp_const.h.

9.27.2.52 #define INTERP_STD 2

Standard interpolation

Definition at line 228 of file fasp_const.h.

9.27.2.53 #define ISPT 2

Isolated points

Definition at line 238 of file fasp_const.h.

9.27.2.54 #define MAT_bBSR 5

block matrix of BSR for bordered systems

Definition at line 110 of file fasp_const.h.

9.27.2.55 #define MAT_bCSR 4

block matrix of CSR

Definition at line 109 of file fasp_const.h.

9.27.2.56 #define MAT_BSR 2

block-wise compressed sparse row

Definition at line 107 of file fasp_const.h.

9.27.2.57 #define MAT_CSR 1

compressed sparse row

Definition at line 106 of file fasp_const.h.

9.27.2.58 #define MAT_CSRL 6

modified CSR to reduce cache missing
Definition at line 111 of file fasp_const.h.

9.27.2.59 #define MAT_FREE 0

Definition of matrix format.
matrix-free format: only mxv action
Definition at line 105 of file fasp_const.h.

9.27.2.60 #define MAT_STR 3

structured sparse matrix
Definition at line 108 of file fasp_const.h.

9.27.2.61 #define MAT_SymCSR 7

symmetric CSR format
Definition at line 112 of file fasp_const.h.

9.27.2.62 #define MAX_AMG_LVL 20

Maximal AMG coarsening level
Definition at line 31 of file fasp_const.h.

9.27.2.63 #define MAX_CRATE 20.0

Maximal coarsening ratio
Definition at line 34 of file fasp_const.h.

9.27.2.64 #define MAX_REFINE_LVL 20

Maximal refinement level
Definition at line 30 of file fasp_const.h.

9.27.2.65 #define MAX_RESTART 20

Maximal number of restarting for BiCGStab
Definition at line 37 of file fasp_const.h.

9.27.2.66 #define MAX_STAG 20

Maximal number of stagnation times

Definition at line 36 of file fasp_const.h.

9.27.2.67 #define MIN_CDOF 20

Minimal number of coarsest variables

Definition at line 32 of file fasp_const.h.

9.27.2.68 #define MIN_CRATE 0.9

Minimal coarsening ratio

Definition at line 33 of file fasp_const.h.

9.27.2.69 #define NL_AMLI_CYCLE 4

Nonlinear AMLI-cycle

Definition at line 193 of file fasp_const.h.

9.27.2.70 #define NO_ORDER 0

Definition of smoothing order.

Natural order smoothing

Definition at line 243 of file fasp_const.h.

9.27.2.71 #define OFF 0

turn off certain parameter

Definition at line 90 of file fasp_const.h.

9.27.2.72 #define ON 1

Definition of switch.

turn on certain parameter

Definition at line 89 of file fasp_const.h.

9.27.2.73 #define OPENMP_HOLDS 2000

Switch to sequence version when size is small

Definition at line 38 of file fasp_const.h.

9.27.2.74 #define PAIRWISE 1

Definition of aggregation types.

pairwise aggregation

Definition at line 184 of file fasp_const.h.

9.27.2.75 #define PREC_AMG 2

with AMG precondition

Definition at line 155 of file fasp_const.h.

9.27.2.76 #define PREC_DIAG 1

with diagonal precondition

Definition at line 154 of file fasp_const.h.

9.27.2.77 #define PREC_FMG 3

with full AMG precondition

Definition at line 156 of file fasp_const.h.

9.27.2.78 #define PREC_ILU 4

with ILU precondition

Definition at line 157 of file fasp_const.h.

9.27.2.79 #define PREC_NULL 0

Definition of preconditioner type for iterative methods.

with no precondition

Definition at line 153 of file fasp_const.h.

9.27.2.80 #define PREC_SCHWARZ 5

with Schwarz preconditioner

Definition at line 158 of file fasp_const.h.

9.27.2.81 #define PRINT_ALL 10

everything: all printouts, including files

Definition at line 100 of file fasp_const.h.

9.27.2.82 #define PRINT_MIN 1

quiet: min info, error, important warnings

Definition at line 96 of file fasp_const.h.

9.27.2.83 #define PRINT_MORE 4

more: print some useful debug information

Definition at line 98 of file fasp_const.h.

9.27.2.84 #define PRINT_MOST 8

most: maximal printouts, no files

Definition at line 99 of file fasp_const.h.

9.27.2.85 #define PRINT_NONE 0

Print level for all subroutines – not including DEBUG output.

silent: no printout at all

Definition at line 95 of file fasp_const.h.

9.27.2.86 #define PRINT_SOME 2

some: more info, less important warnings

Definition at line 97 of file fasp_const.h.

9.27.2.87 #define SA_AMG 2

smoothed aggregation AMG

Definition at line 178 of file fasp_const.h.

9.27.2.88 #define SCHWARZ_BACKWARD 2

Backward ordering

Definition at line 171 of file fasp_const.h.

9.27.2.89 #define SCHWARZ_FORWARD 1

Type of Schwarz smoother.

Forward ordering

Definition at line 170 of file fasp_const.h.

9.27.2.90 #define SCHWARZ_SYMMETRIC 3

Symmetric smoother

Definition at line 172 of file fasp_const.h.

9.27.2.91 #define SMALLREAL 1e-20

A small real number

Definition at line 28 of file fasp_const.h.

9.27.2.92 #define SMALLREAL2 1e-40

An extremely small real number

Definition at line 29 of file fasp_const.h.

9.27.2.93 #define SMOOTHER_BLKOil 11

Definition of specialized smoother types.

Used in monolithic AMG for black-oil

Definition at line 212 of file fasp_const.h.

9.27.2.94 #define SMOOTHER_CG 4

CG as a smoother

Definition at line 201 of file fasp_const.h.

9.27.2.95 #define SMOOTHER_GS 2

Gauss-Seidel smoother

Definition at line 199 of file fasp_const.h.

9.27.2.96 #define SMOOTHER_GSOR 7

GS + SOR smoother

Definition at line 204 of file fasp_const.h.

9.27.2.97 #define SMOOTHER_JACOBI 1

Definition of standard smoother types.

Jacobi smoother

Definition at line 198 of file fasp_const.h.

9.27.2.98 `#define SMOOTHER_L1DIAG 10`

L1 norm diagonal scaling smoother

Definition at line 207 of file fasp_const.h.

9.27.2.99 `#define SMOOTHER_POLY 9`

Polynomial smoother

Definition at line 206 of file fasp_const.h.

9.27.2.100 `#define SMOOTHER_SGS 3`

Symmetric Gauss-Seidel smoother

Definition at line 200 of file fasp_const.h.

9.27.2.101 `#define SMOOTHER_SGSOR 8`

SGS + SSOR smoother

Definition at line 205 of file fasp_const.h.

9.27.2.102 `#define SMOOTHER_SOR 5`

SOR smoother

Definition at line 202 of file fasp_const.h.

9.27.2.103 `#define SMOOTHER_SPETEN 19`

Used in monolithic AMG for black-oil

Definition at line 213 of file fasp_const.h.

9.27.2.104 `#define SMOOTHER_SSOR 6`

SSOR smoother

Definition at line 203 of file fasp_const.h.

9.27.2.105 `#define SOLVER_AMG 21`

AMG as an iterative solver

Definition at line 136 of file fasp_const.h.

9.27.2.106 `#define SOLVER_BiCGstab 2`

Bi-Conjugate Gradient Stabilized

Definition at line 120 of file fasp_const.h.

9.27.2.107 #define SOLVER_CG 1

Conjugate Gradient

Definition at line 119 of file fasp_const.h.

9.27.2.108 #define SOLVER_DEFAULT 0

Definition of solver types for iterative methods.

Use default solver in FASP

Definition at line 117 of file fasp_const.h.

9.27.2.109 #define SOLVER_FMG 22

Full AMG as an solver

Definition at line 137 of file fasp_const.h.

9.27.2.110 #define SOLVER_GCG 7

Generalized Conjugate Gradient

Definition at line 125 of file fasp_const.h.

9.27.2.111 #define SOLVER_GCR 8

Generalized Conjugate Residual

Definition at line 126 of file fasp_const.h.

9.27.2.112 #define SOLVER_GMRES 4

Generalized Minimal Residual

Definition at line 122 of file fasp_const.h.

9.27.2.113 #define SOLVER_MinRes 3

Minimal Residual

Definition at line 121 of file fasp_const.h.

9.27.2.114 #define SOLVER_MUMPS 33

MUMPS Direct Solver

Definition at line 141 of file fasp_const.h.

9.27.2.115 `#define SOLVER_SBiCGstab 12`

BiCGstab with safe net

Definition at line 129 of file fasp_const.h.

9.27.2.116 `#define SOLVER_SCG 11`

Conjugate Gradient with safe net

Definition at line 128 of file fasp_const.h.

9.27.2.117 `#define SOLVER_SGCG 17`

GCG with safe net

Definition at line 134 of file fasp_const.h.

9.27.2.118 `#define SOLVER_SGMRES 14`

GMRes with safe net

Definition at line 131 of file fasp_const.h.

9.27.2.119 `#define SOLVER_SMinRes 13`

MinRes with safe net

Definition at line 130 of file fasp_const.h.

9.27.2.120 `#define SOLVER_SUPERLU 31`

SuperLU Direct Solver

Definition at line 139 of file fasp_const.h.

9.27.2.121 `#define SOLVER_SVFGMRES 16`

Variable-restart FGMRES with safe net

Definition at line 133 of file fasp_const.h.

9.27.2.122 `#define SOLVER_SVGMRES 15`

Variable-restart GMRES with safe net

Definition at line 132 of file fasp_const.h.

9.27.2.123 `#define SOLVER_UMFPACK 32`

UMFPack Direct Solver

Definition at line 140 of file fasp_const.h.

9.27.2.124 #define SOLVER_VFGMRES 6

Variable Restarting Flexible GMRES

Definition at line 124 of file fasp_const.h.

9.27.2.125 #define SOLVER_VGMRES 5

Variable Restarting GMRES

Definition at line 123 of file fasp_const.h.

9.27.2.126 #define STAG_RATIO 1e-4

Stagnation tolerance = tol*STAGRATIO

Definition at line 35 of file fasp_const.h.

9.27.2.127 #define STOP_MOD_REL_RES 3

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

Definition at line 148 of file fasp_const.h.

9.27.2.128 #define STOP_REL_PRECRES 2

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

Definition at line 147 of file fasp_const.h.

9.27.2.129 #define STOP_REL_RES 1

Definition of iterative solver stopping criteria types.

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

Definition at line 146 of file fasp_const.h.

9.27.2.130 #define TRUE 1

Definition of logic type.

logic TRUE

Definition at line 83 of file fasp_const.h.

9.27.2.131 #define UA_AMG 3

unsmoothed aggregation AMG

Definition at line 179 of file fasp_const.h.

9.27.2.132 `#define UNPT -1`

Undetermined points

Definition at line 235 of file `fasp_const.h`.

9.27.2.133 `#define USERDEFINED 0`

Type of ordering for smoothers.

User defined order

Definition at line 249 of file `fasp_const.h`.

9.27.2.134 `#define V_CYCLE 1`

Definition of cycle types.

V-cycle

Definition at line 190 of file `fasp_const.h`.

9.27.2.135 `#define VMB 2`

VMB aggregation

Definition at line 185 of file `fasp_const.h`.

9.27.2.136 `#define W_CYCLE 2`

W-cycle

Definition at line 191 of file `fasp_const.h`.

9.28 `fmgcycle.c` File Reference

Abstract non-recursive full multigrid cycle.

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

Functions

- void `fasp_solver_fmgcycle` (`AMG_data *mgl`, `AMG_param *param`)

Solve $Ax=b$ with non-recursive full multigrid K -cycle.

9.28.1 Detailed Description

Abstract non-recursive full multigrid cycle.

9.28.2 Function Documentation

9.28.2.1 void fasp_solver_fmecycle (AMG_data * mgl, AMG_param * param)

Solve $Ax=b$ with non-recursive full multigrid K-cycle.

Parameters

<i>mgl</i>	Pointer to AMG data: 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.

Definition at line 35 of file fmecycle.c.

9.29 formats.c File Reference

Matrix format conversion routines.

```
#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_dcsr_dcsr](#) (dCSRmat *A)
Convert a dCSRmat into a dCSRmat.
- [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.

9.29.1 Detailed Description

Matrix format conversion routines.

9.29.2 Function Documentation

9.29.2.1 `dCSRmat fasp_format_bdcsr_dcsr (block_dCSRmat * Ab)`

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

Parameters

<i>Ab</i>	Pointer to <i>block_dCSRmat</i> matrix
-----------	--

Returns

dCSRmat matrix if succeed, NULL if fail

Author

Shiquan Zhang

Date

08/10/2010

Definition at line 292 of file formats.c.

9.29.2.2 `dCOOmat * fasp_format_dbsr_dcoo (dBSRmat * B)`

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

Parameters

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

Returns

Pointer to *dCOOmat* matrix

Author

Zhiyang Zhou

Date

2010/10/26

Definition at line 944 of file formats.c.

9.29.2.3 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.

9.29.2.4 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.

9.29.2.5 `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.

9.29.2.6 **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.

9.29.2.7 [dCSRLmat](#) * fasp_format_dcsrl_dcsr ([dCSRmat](#) * *A*)

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

Parameters

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

Returns

Pointer to [dCSRmat](#) matrix

Author

Zhiyang Zhou

Date

2011/01/07

Definition at line 361 of file formats.c.

9.29.2.8 [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 840 of file formats.c.

9.29.2.9 [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.

9.30 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 $y \mid \beta e_1 - H * y \mid$.

9.30.1 Detailed Description

Givens transformation.

9.30.2 Function Documentation

9.30.2.1 void [fasp_aux_givens](#) (const [REAL](#) *beta*, [dCSRmat](#) * *H*, [dvector](#) * *y*, [REAL](#) * *tmp*)

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

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

Author

Xuehai Huang

Date

10/19/2008

Definition at line 28 of file givens.c.

9.31 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** prtvl)

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** prtvl)

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** prtvl)

Solve $Ax=b$ of Poisson 3D equation by Geometric Multigrid Method.
- void **fasp_poisson_fgmg_1D** (**REAL** *u, **REAL** *b, **INT** nx, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

Solve $Ax=b$ of Poisson 1D equation by Geometric Multigrid Method (Full Multigrid)
- void **fasp_poisson_fgmg_2D** (**REAL** *u, **REAL** *b, **INT** nx, **INT** ny, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

Solve $Ax=b$ of Poisson 2D equation by Geometric Multigrid Method (Full Multigrid)
- void **fasp_poisson_fgmg_3D** (**REAL** *u, **REAL** *b, **INT** nx, **INT** ny, **INT** nz, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

Solve $Ax=b$ of Poisson 3D equation by Geometric Multigrid Method (Full Multigrid)
- **INT fasp_poisson_pcg_gmg_1D** (**REAL** *u, **REAL** *b, **INT** nx, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

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, **INT** nx, **INT** ny, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

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, **INT** nx, **INT** ny, **INT** nz, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

Solve $Ax=b$ of Poisson 3D equation by Geometric Multigrid Method (GMG preconditioned Conjugate Gradient method)

9.31.1 Detailed Description

GMG method as an iterative solver for Poisson Problem.

9.31.2 Function Documentation

9.31.2.1 void fasp_poisson_fgmg_1D (**REAL** * u, **REAL** * b, **INT** nx, **INT** maxlevel, **REAL** rtol, const **SHORT** prtvl)

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.

9.31.2.2 void fasp_poisson_fgmg_2D (REAL * *u*, REAL * *b*, INT *nx*, INT *ny*, INT *maxlevel*, 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.

9.31.2.3 void fasp_poisson_fgmg_3D (REAL * *u*, REAL * *b*, INT *nx*, INT *ny*, INT *nz*, INT *maxlevel*, 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 631 of file gmg_poisson.c.

9.31.2.4 **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.

9.31.2.5 **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.

9.31.2.6 `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.

9.31.2.7 INT fasp_poisson_pcg_gmg_1D (REAL * *u*, REAL * *b*, INT *nx*, INT *maxlevel*, 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 740 of file gmg_poisson.c.

9.31.2.8 `INT fasp_poisson_pcg_gmg_2D (REAL * u, REAL * b, INT nx, INT ny, INT maxlevel, 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 833 of file gmg_poisson.c.

9.31.2.9 `INT fasp_poisson_pcg_gmg_3D (REAL * u, REAL * b, INT nx, INT ny, INT nz, INT maxlevel, 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 941 of file gmg_poisson.c.

9.32 graphics.c File Reference

Functions 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.

9.32.1 Detailed Description

Functions for graphical output.

9.32.2 Function Documentation

9.32.2.1 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 462 of file graphics.c.

9.32.2.2 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.

9.32.2.3 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 623 of file graphics.c.

9.32.2.4 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>	size*size 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.

9.32.2.5 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.

9.33 ilu.f File Reference

ILU routines for preconditioning adapted from SPARSEKIT.

Functions/Subroutines

- subroutine **iluk** (n, a, ja, ia, lfil, alu, jlu, iwk, ierr, nzlu)
- subroutine **ilut** (n, a, ja, ia, lfil, droptol, alu, jlu, iwk, ierr, nz)
- subroutine **ilutp** (n, a, ja, ia, lfil, droptol, permtol, mbloc, alu, jlu, iwk, ierr, nz)
- subroutine **srtr** (num, q)
- subroutine **qspllit** (a, ind, n, ncut)
- subroutine **symbfactor** (n, colind, rwptr, levfill, nzmax, nzlu, ijlu, uptr, ierr)

9.33.1 Detailed Description

ILU routines for preconditioning adapted from SPARSEKIT.

Note

Incomplete Factorization Methods: ILUk, ILUt, ILUtp

9.34 ilu_setup_bsr.c File ReferenceSetup 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.

9.34.1 Detailed Description

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

9.34.2 Function Documentation

9.34.2.1 [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](#).

9.35 ilu_setup_csr.c File Reference

Setup of ILU 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.

9.35.1 Detailed Description

Setup of ILU decomposition for [dCSRmat](#) matrices.

9.35.2 Function Documentation

9.35.2.1 [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](#).

9.36 [ilu_setup_str.c](#) File Reference

Setup of ILU decomposition for [dSTRmat](#) matrices.

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


Functions

- void `fasp_ilu_dstr_setup0` (`dSTRmat *A`, `dSTRmat *LU`)
Get $ILU(0)$ decomposition of a structured matrix A .
- void `fasp_ilu_dstr_setup1` (`dSTRmat *A`, `dSTRmat *LU`)
Get $ILU(1)$ decomposition of a structured matrix A .

9.36.1 Detailed Description

Setup of ILU decomposition for `dSTRmat` matrices.

9.36.2 Function Documentation

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

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

Parameters

A	Pointer to <code>dSTRmat</code>
LU	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`.

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

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

Parameters

A	Pointer to original structured matrix of REAL type
LU	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.

9.37 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](#) ([INT](#) iwk, [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 precondition data.

9.37.1 Detailed Description

Initialize important data structures.

Note

Every structures should be initialized before usage.

9.37.2 Function Documentation

9.37.2.1 `AMG_data_bsr * fasp_amg_data_bsr_create (SHORT max_levels)`

Create and initialize [AMG_data](#) data sturcture for AMG/SAMG (BSR format)

Parameters

<i>max_levels</i>	Max number of levels allowed
-------------------	------------------------------

Returns

Pointer to the [AMG_data](#) data structure

Author

Xiaozhe Hu

Date

08/07/2011

Definition at line 85 of file init.c.

9.37.2.2 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 243 of file init.c.

9.37.2.3 [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 55 of file init.c.

9.37.2.4 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!

Definition at line 185 of file init.c.

9.37.2.5 void fasp_ilu_data_alloc (INT *iwk*, 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 116 of file init.c.

9.37.2.6 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 289 of file init.c.

9.37.2.7 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 310 of file init.c.

9.37.2.8 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 24 of file init.c.

9.37.2.9 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 326 of file init.c.

9.37.2.10 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 145 of file init.c.

9.38 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.

9.38.1 Detailed Description

Read input parameters.

9.38.2 Function Documentation

9.38.2.1 [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.

9.38.2.2 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.

9.39 interface_mumps.c File Reference

Interface to MUMPS direct solvers.

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

Functions

- int [fasp_solver_mumps](#) (dCSRmat *ptrA, dvector *b, dvector *u, const SHORT prtlvl)
Solve $Ax=b$ by MUMPS directly.
- int [fasp_solver_mumps_steps](#) (dCSRmat *ptrA, dvector *b, dvector *u, Mumps_data *mumps)
Solve $Ax=b$ by MUMPS in three steps.

9.39.1 Detailed Description

Interface to MUMPS direct solvers.

9.39.2 Function Documentation

9.39.2.1 `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 35 of file interface_mumps.c.

9.39.2.2 `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 163 of file interface_mumps.c.

9.40 interface_samg.c File Reference

Interface to SAMG.

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

Functions

- void [dvector2SAMGInput](#) ([dvector](#) *vec, char *filename)
Write a dvector to disk file in SAMG format (coordinate format)
- [INT dCSRmat2SAMGInput](#) ([dCSRmat](#) *A, char *filefrm, char *fileamg)
Write SAMG Input data from a sparse matrix of CSR format.

9.40.1 Detailed Description

Interface to SAMG.

Add reference for SAMG by K. Stuben here!

9.40.2 Function Documentation

9.40.2.1 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>	pointer to the name of the .frm file
<i>fileamg</i>	pointer to the name of the .amg file

Author

Zhiyang Zhou

Date

2010/08/25

Definition at line 56 of file interface_samg.c.

9.40.2.2 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>	char for vector file name

Author

Zhiyang Zhou

Date

08/25/2010

Definition at line 27 of file interface_samg.c.

9.41 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.

9.41.1 Detailed Description

Interface to SuperLU direct solvers.

9.41.2 Function Documentation

9.41.2.1 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 39 of file interface_superlu.c.

9.42 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.

9.42.1 Detailed Description

Interface to UMFPACK direct solvers.

9.42.2 Function Documentation

9.42.2.1 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 34 of file interface_umfpack.c.

9.43 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.

9.43.1 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.

9.43.2 Function Documentation

9.43.2.1 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 <code>dCSRmat</code> : 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`.

9.43.2.2 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.

9.43.2.3 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.

9.44 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.

9.44.1 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

9.44.2 Function Documentation

9.44.2.1 void `fasp_amg_interp_em` (`dCSRmat` * A, `ivector` * vertices, `dCSRmat` * P, `AMG_param` * param)

Energy-min interpolation.

Parameters

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

9.45 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 *filename, 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 *filename, 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

9.45.1 Detailed Description

Matrix-vector input/output subroutines.

Note

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

9.45.2 Function Documentation

9.45.2.1 void fasp_dbsr_print (`dBSRmat` * A)

Print out a `dBSRmat` matrix in coordinate format.

Parameters

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

Author

Ziteng Wang

Date

12/24/2012

Modified by Chunsheng Feng on 11/16/2013

Definition at line 1444 of file io.c.

9.45.2.2 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 <i>A</i>
<i>A</i>	Pointer to the dBSRmat <i>A</i>

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.

9.45.2.3 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.

9.45.2.4 `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 A

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.

9.45.2.5 `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.

9.45.2.6 void fasp_dcoo_print (dCOOmat * A)

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

Parameters

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

Author

Ziteng Wang

Date

12/24/2012

Definition at line 1423 of file io.c.

9.45.2.7 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.

9.45.2.8 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.

9.45.2.9 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.

9.45.2.10 void fasp_dcsr_print (dCSRmat * A)

Print out a dCSRmat matrix in coordinate format.

Parameters

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

Author

Xuehai Huang

Date

03/29/2009

Definition at line 1401 of file io.c.

9.45.2.11 void fasp_dcsr_read (const char * filename, dCSRmat * A)

Read A from matrix disk file in IJ format.

Parameters

*filename	char for matrix file name
*A	pointer to the CSR matrix

Author

Ziteng Wang

Date

12/25/2012

Definition at line 257 of file io.c.

9.45.2.12 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 <i>A</i>

Author

Chunsheng Feng

Date

11/14/2013

Definition at line 1531 of file io.c.

9.45.2.13 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.

9.45.2.14 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.

9.45.2.15 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.

9.45.2.16 `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.

9.45.2.17 `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 [dCSR↔Rmat](#) format. For details of mtx format, please refer to <http://math.nist.gov/MatrixMarket/>.
Indices start from 1, NOT 0!!!

Author

Chensong Zhang

Date

09/05/2011

Definition at line 472 of file io.c.

9.45.2.18 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.

9.45.2.19 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 1571 of file io.c.

9.45.2.20 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.

9.45.2.21 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.

9.45.2.22 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 <i>n</i> =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.

9.45.2.23 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 <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

Chensong Zhang

Date

03/29/2009

Definition at line 810 of file io.c.

9.45.2.24 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.

9.45.2.25 void fasp_dvecind_read (const char * *filename*, dvector * *b*)

Read b from matrix disk file.

Parameters

<i>filename</i>	File name for vector b
<i>b</i>	Pointer to the dvector b (output)

Note

File Format:

- nrow
- ind_j, val_j, j=0:nrow-1

Because the index is given, order is not important!

Author

Chensong Zhang

Date

03/29/2009

Definition at line 760 of file io.c.

9.45.2.26 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.

9.45.2.27 `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

Xiaohe Hu

Date

05/30/2014

Definition at line 2062 of file io.c.

9.45.2.28 `void fasp_ivec_print (INT n, ivec * u)`

Print first n entries of a vector of INT type.

Parameters

<i>n</i>	An interger (if n=0, then print all entries)
<i>u</i>	Pointer to an ivector

Author

Chensong Zhang

Date

03/29/2009

Definition at line 1382 of file io.c.

9.45.2.29 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 b
<i>b</i>	Pointer to the dvector b (output)

Note

File Format:

- nrow
- val_j, j=0:nrow-1

Author

Xuehai Huang

Date

03/29/2009

Definition at line 902 of file io.c.

9.45.2.30 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.

9.45.2.31 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.

9.45.2.32 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 1605 of file io.c.

9.45.2.33 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 1710 of file io.c.

9.45.2.34 `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 1784 of file io.c.

9.45.2.35 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 1877 of file io.c.

9.45.2.36 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 1974 of file io.c.

9.45.3 Variable Documentation

9.45.3.1 INT dlength

Length of REAL in byte

Definition at line 14 of file io.c.

9.45.3.2 INT ilength

Length of INT in byte

Definition at line 13 of file io.c.

9.46 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.

9.46.1 Detailed Description

Iterative solvers for [block_dCSRmat](#) matrices.

9.46.2 Function Documentation

9.46.2.1 **INT** fasp_solver_bdcsr_itsolver (**block_dCSRmat** * *A*, **dvector** * *b*, **dvector** * *x*, **precond** * *pc*, **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>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.

9.46.2.2 **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.

9.46.2.3 **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.

9.46.2.4 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.

9.46.2.5 **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.

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.

9.47 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.

9.47.1 Detailed Description

Iterative solvers for `dBSRmat` matrices.

9.47.2 Function Documentation

9.47.2.1 `INT fasp_solver_dbsr_itsolver (dBSRmat * A, dvector * b, dvector * x, precondition * pc, itsolver_param * itparam)`

Solve $Ax=b$ by preconditioned Krylov methods for BSR matrices.

Parameters

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

Returns

Iteration number if converges; ERROR otherwise.

Author

Zhiyang Zhou, Xiaozhe Hu

Date

10/26/2010

Definition at line 37 of file `itsolver_bsr.c`.

9.47.2.2 `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.

9.47.2.3 `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.

9.47.2.4 `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.

9.47.2.5 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.

9.47.2.6 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.

9.47.2.7 `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 644 of file itsolver_bsr.c.

9.48 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.

9.48.1 Detailed Description

Iterative solvers for [dCSRmat](#) matrices.

9.48.2 Function Documentation

9.48.2.1 [INT fasp_solver_dcsr_itsolver](#) ([dCSRmat](#) * A, [dvector](#) * b, [dvector](#) * x, [precond](#) * pc, [itsolver_param](#) * itparam)

Solve $Ax=b$ by preconditioned Krylov methods for CSR matrices.

Parameters

<i>A</i>	Pointer to the coeff matrix in 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>pc</i>	Pointer to the preconditioning action
<i>itparam</i>	Pointer to parameters for iterative solvers

Returns

Iteration number if converges; ERROR otherwise.

Author

Chensong Zhang

Date

09/25/2009

Note

This is an abstract interface for iterative methods.

Definition at line 39 of file itsolver_csr.c.

9.48.2.2 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.

9.48.2.3 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.

9.48.2.4 `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.

9.48.2.5 `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.

9.48.2.6 `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.

9.48.2.7 `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.

9.48.2.8 `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.

9.49 itsolver_mf.c File Reference

Iterative solvers with matrix-free spmv.

```
#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.

9.49.1 Detailed Description

Iterative solvers with matrix-free spmv.

9.49.2 Function Documentation

9.49.2.1 [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.

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.

9.49.2.2 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.

9.49.2.3 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.

9.50 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.

9.50.1 Detailed Description

Iterative solvers for [dSTRmat](#) matrices.

9.50.2 Function Documentation

9.50.2.1 [INT fasp_solver_dstr_itsolver](#) ([dSTRmat](#) * A, [dvector](#) * b, [dvector](#) * x, [precond](#) * 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.

9.50.2.2 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.

9.50.2.3 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.

9.50.2.4 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.

9.50.2.5 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.

9.51 lu.c File Reference

LU decomposition and direct solve for dense matrix.

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

Functions

- [SHORT fasp_smat_lu_decomp](#) ([REAL *A](#), [INT pivot\[\]](#), [INT n](#))
LU decomposition of A using Doolittle's method.
- [SHORT fasp_smat_lu_solve](#) ([REAL *A](#), [REAL b\[\]](#), [INT pivot\[\]](#), [REAL x\[\]](#), [INT n](#))
Solving $Ax=b$ using LU decomposition.

9.51.1 Detailed Description

LU decomposition and direct solve for dense matrix.

9.51.2 Function Documentation

9.51.2.1 SHORT fasp_smat_lu_decomp (REAL * A, INT pivot[], INT n)

LU decomposition of A using Doolittle's method.

Parameters

<i>A</i>	Pointer to the full matrix
<i>pivot</i>	Pivoting positions
<i>n</i>	Size of matrix A

Returns

FASP_SUCCESS if succeeded; otherwise, error information.

Note

Use Doolittle's method to decompose the $n \times n$ matrix A into a unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$. The matrices L and U replace the matrix A . The diagonal elements of L are 1 and are not stored.

The Doolittle method with partial pivoting is: Determine the pivot row and interchange the current row with the pivot row, then assuming that row k is the current row, $k = 0, \dots, n - 1$ evaluate in order the following pair of expressions $U[k][j] = A[k][j] - (L[k][0]*U[0][j] + \dots + L[k][k-1]*U[k-1][j])$ for $j = k, k+1, \dots, n-1$ $L[i][k] = (A[i][k] - (L[i][0]*U[0][k] + \dots + L[i][k-1]*U[k-1][k])) / U[k][k]$ for $i = k+1, \dots, n-1$.

Author

Xuehai Huang

Date

04/02/2009

Definition at line 46 of file lu.c.

9.51.2.2 SHORT fasp_smat_lu_solve (REAL * A , REAL $b[]$, INT $pivot[]$, REAL $x[]$, INT n)

Solving $Ax=b$ using LU decomposition.

Parameters

A	Pointer to the full matrix
b	Right hand side array
$pivot$	Pivoting positions
x	Pointer to the solution array
n	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.

9.52 memory.c File Reference

Memory allocation and deallocation.

```
#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.

9.52.1 Detailed Description

Memory allocation and deallocation.

9.52.2 Function Documentation

9.52.2.1 void * [fasp_mem_calloc](#) (LONGLONG size, INT type)

1M = 1024*1024

Allocate, initiate, and check memory

Parameters

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

Returns

Void pointer to the allocated memory

Author

Chensong Zhang

Date

2010/08/12

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

Definition at line 58 of file memory.c.

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

Check wether 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 192 of file memory.c.

9.52.2.3 **SHORT** fasp_mem_dcsr_check (dCSRmat * *A*)

Check wether a [dCSRmat](#) *A* has sucessfully 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 242 of file memory.c.

9.52.2.4 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 145 of file memory.c.

9.52.2.5 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 216 of file memory.c.

9.52.2.6 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 111 of file memory.c.

9.52.2.7 void fasp_mem_usage ()

Show total allocated memory currently.

Author

Chensong Zhang

Date

2010/08/12

Definition at line 170 of file memory.c.

9.52.3 Variable Documentation**9.52.3.1 unsigned INT total_alloc_count = 0**

Total allocated memory amount.

total allocation times

Definition at line 33 of file memory.c.

9.52.3.2 unsigned INT total_alloc_mem = 0

total allocated memory

Definition at line 32 of file memory.c.

9.53 message.c File Reference

Output some useful messages.

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

9.53.1 Detailed Description

Output some useful messages.

Note

These routines are meant for internal use only.

9.53.2 Function Documentation

9.53.2.1 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.

9.53.2.2 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.

9.53.2.3 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.

9.53.2.4 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.

9.53.2.5 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.

9.53.2.6 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.

9.54 mgcycle.c File Reference

Abstract non-recursive multigrid cycle.

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

Functions

- void [fasp_solver_mgcycle](#) ([AMG_data](#) *mgl, [AMG_param](#) *param)
Solve $Ax=b$ with non-recursive multigrid cycle.
- void [fasp_solver_mgcycle_bsr](#) ([AMG_data_bsr](#) *mgl, [AMG_param](#) *param)
Solve $Ax=b$ with non-recursive multigrid cycle.

9.54.1 Detailed Description

Abstract non-recursive multigrid cycle.

9.54.2 Function Documentation

9.54.2.1 void [fasp_solver_mgcycle](#) ([AMG_data](#) * *mgl*, [AMG_param](#) * *param*)

Solve $Ax=b$ with non-recursive multigrid cycle.

Parameters

<i>mgl</i>	Pointer to AMG data: 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.

9.54.2.2 void fasp_solver_mgcycle_bsr (AMG_data_bsr * *mgl*, AMG_param * *param*)

Solve $Ax=b$ with non-recursive multigrid cycle.

Parameters

<i>mgI</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 257 of file mgcycle.c.

9.55 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](#) **mgI*, [AMG_param](#) **param*, INT *level*)
Solve $Ax=b$ with recursive multigrid K-cycle.

9.55.1 Detailed Description

Abstract multigrid cycle – recursive version.

Note

Not used any more. Will be removed! –Chensong

9.55.2 Function Documentation

9.55.2.1 void [fasp_solver_mgrecur](#) ([AMG_data](#) * *mgI*, [AMG_param](#) * *param*, INT *level*)

Solve $Ax=b$ with recursive multigrid K-cycle.

Parameters

<i>mgI</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.

9.56 ordering.c File Reference

A collection of ordering, merging, removing duplicated integers functions.

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

Functions

- [INT fasp_BinarySearch](#) ([INT](#) *list, [INT](#) value, [INT](#) nlist)
Binary Search.
- [INT fasp_aux_unique](#) ([INT](#) numbers[], [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)
Resverse CMK ordering.

9.56.1 Detailed Description

A collection of ordering, merging, removing duplicated integers functions.

9.56.2 Function Documentation

9.56.2.1 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.

9.56.2.2 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.

9.56.2.3 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.

9.56.2.4 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.

9.56.2.5 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.

9.56.2.6 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.

9.56.2.7 INT fasp_aux_unique (INT *numbers*[], 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.

9.56.2.8 INT fasp_BinarySearch (INT * *list*, INT *value*, 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.

9.56.2.9 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.

9.56.2.10 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.

9.57 parameters.c File Reference

Initialize, set, or print input data and parameters.

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

Functions

- void [fasp_param_set](#) (int argc, const char *argv[], [input_param](#) *iniparam)
Read input from command-line arguments.

- void `fasp_param_init` (`input_param` *iniparam, `itsolver_param` *itsparam, `AMG_param` *amgparam, `ILU_param` *iluparam, `Schwarz_param` *schparam)
Initialize parameters, global variables, etc.
- void `fasp_param_input_init` (`input_param` *iniparam)
Initialize input parameters.
- void `fasp_param_amg_init` (`AMG_param` *amgparam)
Initialize AMG parameters.
- void `fasp_param_solver_init` (`itsolver_param` *itsparam)
Initialize itsolver_param.
- void `fasp_param_ilu_init` (`ILU_param` *iluparam)
Initialize ILU parameters.
- void `fasp_param_Schwarz_init` (`Schwarz_param` *schparam)
Initialize Schwarz parameters.
- void `fasp_param_amg_set` (`AMG_param` *param, `input_param` *iniparam)
Set AMG_param from INPUT.
- void `fasp_param_ilu_set` (`ILU_param` *iluparam, `input_param` *iniparam)
Set ILU_param with INPUT.
- void `fasp_param_Schwarz_set` (`Schwarz_param` *schparam, `input_param` *iniparam)
Set Schwarz_param with INPUT.
- void `fasp_param_solver_set` (`itsolver_param` *itsparam, `input_param` *iniparam)
Set itsolver_param with INPUT.
- void `fasp_param_amg_to_prec` (`precond_data` *pcdata, `AMG_param` *amgparam)
Set precondition_data with AMG_param.
- void `fasp_param_prec_to_amg` (`AMG_param` *amgparam, `precond_data` *pcdata)
Set AMG_param with precondition_data.
- void `fasp_param_amg_to_prec_bsr` (`precond_data_bsr` *pcdata, `AMG_param` *amgparam)
Set precondition_data_bsr with AMG_param.
- void `fasp_param_prec_to_amg_bsr` (`AMG_param` *amgparam, `precond_data_bsr` *pcdata)
Set AMG_param with precondition_data.
- void `fasp_param_amg_print` (`AMG_param` *param)
Print out AMG parameters.
- void `fasp_param_ilu_print` (`ILU_param` *param)
Print out ILU parameters.
- void `fasp_param_Schwarz_print` (`Schwarz_param` *param)
Print out Schwarz parameters.
- void `fasp_param_solver_print` (`itsolver_param` *param)
Print out itsolver parameters.

9.57.1 Detailed Description

Initialize, set, or print input data and parameters.

9.57.2 Function Documentation

9.57.2.1 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.

9.57.2.2 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.

9.57.2.3 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.

9.57.2.4 void fasp_param_amg_to_prec (precondition_data * pcddata, 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.

9.57.2.5 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.

9.57.2.6 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.

9.57.2.7 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.

9.57.2.8 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.

9.57.2.9 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.

9.57.2.10 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.

9.57.2.11 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.

9.57.2.12 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.

9.57.2.13 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.

9.57.2.14 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.

9.57.2.15 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.

9.57.2.16 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.

9.57.2.17 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.

9.57.2.18 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.

9.57.2.19 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.

9.58 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 $Au=b$.

9.58.1 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

9.58.2 Function Documentation

9.58.2.1 `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.

9.58.2.2 `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.

9.58.2.3 `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.

9.58.2.4 `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.

9.59 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 $Au=b$.

9.59.1 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 $r_0 = b - A \cdot x_0$ and convergence check;

Step 2. Initialization $z_0 = M^{-1} \cdot r_0, p_0 = z_0$;

Step 3. Main loop ...

FOR $k = 0 : \text{MaxIt}$

- get step size $\alpha = f(r_k, z_k, p_k)$;
- update solution: $x_{k+1} = x_k + \alpha \cdot p_k$;
- perform stagnation check;
- update residual: $r_{k+1} = r_k - \alpha \cdot (A \cdot p_k)$;
- 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 $\text{norm}(\alpha \cdot p_k)/\text{norm}(x_{k+1}) < \text{tol_stag}$
 1. compute $r = b - A \cdot x_{k+1}$;
 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 $r = b - A \cdot x_{k+1}$;
 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

9.59.2 Function Documentation

9.59.2.1 `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.

9.60 pcg.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient.

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

Functions

- [INT fasp_solver_dcsr_pcg](#) ([dCSRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [precond](#) *pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop_type, const [SHORT](#) prtlvl)
Preconditioned conjugate gradient method for solving $Au=b$.
- [INT fasp_solver_dbsr_pcg](#) ([dBSRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [precond](#) *pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop_type, const [SHORT](#) prtlvl)
Preconditioned conjugate gradient method for solving $Au=b$.
- [INT fasp_solver_bdcsr_pcg](#) ([block_dCSRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [precond](#) *pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop_type, const [SHORT](#) prtlvl)
Preconditioned conjugate gradient method for solving $Au=b$.
- [INT fasp_solver_dstr_pcg](#) ([dSTRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [precond](#) *pc, const [REAL](#) tol, const [INT](#) MaxIt, const [SHORT](#) stop_type, const [SHORT](#) prtlvl)
Preconditioned conjugate gradient method for solving $Au=b$.

9.60.1 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

9.60.2 Function Documentation

9.60.2.1 `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 656 of file pcg.c.

9.60.2.2 `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 369 of file pcg.c.

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

Preconditioned conjugate gradient method for solving $Au=b$.

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.

9.60.2.4 `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 943 of file pcg.c.

9.61 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 $Au=b$.

9.61.1 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

9.61.2 Function Documentation

9.61.2.1 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.

9.62 pgcg.c File Reference

Krylov subspace methods – Preconditioned Generalized CG.

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

Functions

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

Preconditioned genirilzed conjugate gradient (GCG) method for solving $Au=b$.

9.62.1 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

9.62.2 Function Documentation

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

Preconditioned genirilzed conjugate gradient (GCG) method for solving $Au=b$.

Parameters

<i>A</i>	Pointer to 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.

9.63 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 $Au=b$.

9.63.1 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

9.63.2 Function Documentation

9.63.2.1 [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.

9.64 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_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 $Au=b$.
- [INT fasp_solver_dcsr_pgcr](#) (dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)
A preconditioned GCR method for solving $Au=b$.

9.64.1 Detailed Description

Krylov subspace methods – Preconditioned GCR.

9.64.2 Function Documentation

9.64.2.1 `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 250 of file pgcr.c.

9.64.2.2 `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

Definition at line 37 of file pgcr.c.

9.65 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, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)
Right preconditioned GMRES method for solving $Au=b$.
- **INT fasp_solver_bdcsr_pgmres** (block_dCSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)
Preconditioned GMRES method for solving $Au=b$.
- **INT fasp_solver_dbsr_pgmres** (dBSRmat *A, dvector *b, dvector *x, precondition *pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT prtlvl)
Preconditioned GMRES method for solving $Au=b$.
- **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 $Au=b$.

9.65.1 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

9.65.2 Function Documentation

9.65.2.1 INT fasp_solver_bdcsr_pgmres (block_dCSRmat * A, dvector * b, dvector * x, precondition * pc, const REAL tol, const INT MaxIt, const SHORT restart, const SHORT stop_type, const SHORT 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 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.

9.65.2.2 `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.

9.65.2.3 `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.

9.65.2.4 `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.

9.66 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.

9.66.1 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.

9.66.2 Function Documentation

9.66.2.1 **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.

9.67 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, precondition *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, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtvl)`

A preconditioned minimal residual (Minres) method for solving $Au=b$.

9.67.1 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

9.67.2 Function Documentation

9.67.2.1 `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.

9.67.2.2 `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.

9.67.2.3 `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.

9.68 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, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl)`
A preconditioned minimal residual (Minres) method for solving $Au=b$.

9.68.1 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

9.68.2 Function Documentation

9.68.2.1 **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 $Au=b$.

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.

9.69 precondition_bcsr.c File Reference

Preconditioners for block CSR matrices.

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

Functions

- void [fasp_precond_block_diag_3](#) (double *r, double *z, void *data)
block diagonal preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_diag_3_amg](#) (double *r, double *z, void *data)
block diagonal preconditioning (3x3 block matrix, each diagonal block is solved by AMG)
- void [fasp_precond_block_diag_4](#) (double *r, double *z, void *data)
block diagonal preconditioning (4x4 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_lower_3](#) (double *r, double *z, void *data)
block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_lower_3_amg](#) (double *r, double *z, void *data)
block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved by AMG)
- void [fasp_precond_block_lower_4](#) (double *r, double *z, void *data)
block lower triangular preconditioning (4x4 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_upper_3](#) (double *r, double *z, void *data)
block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_upper_3_amg](#) (double *r, double *z, void *data)
block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved AMG)
- void [fasp_precond_block_SGS_3](#) (double *r, double *z, void *data)
block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_block_SGS_3_amg](#) (double *r, double *z, void *data)
block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void [fasp_precond_sweeping](#) (double *r, double *z, void *data)
sweeping preconditioner for Maxwell equations

9.69.1 Detailed Description

Preconditioners for block CSR matrices.

9.69.2 Function Documentation

9.69.2.1 `void fasp_precond_block_diag_3 (double * r, double * 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.

9.69.2.2 void fasp_precond_block_diag_3_amg (double * *r*, double * *z*, void * *data*)

block diagonal preconditioning (3x3 block matrix, each diagonal block is solved by AMG)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

07/10/2014

Definition at line 101 of file precondition_bcsr.c.

9.69.2.3 void fasp_precond_block_diag_4 (double * *r*, double * *z*, void * *data*)

block diagonal preconditioning (4x4 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

07/10/2014

Definition at line 166 of file precondition_bcsr.c.

9.69.2.4 `void fasp_precond_block_lower_3 (double * r, double * z, void * data)`

block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

07/10/2014

Definition at line 252 of file precondition_bcsr.c.

9.69.2.5 void fasp_precond_block_lower_3_amg (double * *r*, double * *z*, void * *data*)

block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved by AMG)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

07/10/2014

Definition at line 334 of file precondition_bcsr.c.

9.69.2.6 void fasp_precond_block_lower_4 (double * *r*, double * *z*, void * *data*)

block lower triangular preconditioning (4x4 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

07/10/2014

Definition at line 408 of file precondition_bcsr.c.

9.69.2.7 void fasp_precond_block_SGS_3 (double * *r*, double * *z*, void * *data*)

block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

02/19/2015

Definition at line 669 of file precondition_bcsr.c.

9.69.2.8 void fasp_precond_block_SGS_3_amg (double * *r*, double * *z*, void * *data*)

block symmetric GS preconditioning (3x3 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

02/19/2015

Definition at line 788 of file precondition_bcsr.c.

9.69.2.9 void fasp_precond_block_upper_3 (double * *r*, double * *z*, void * *data*)

block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

02/18/2015

Definition at line 506 of file precondition_bcsr.c.

9.69.2.10 void fasp_precond_block_upper_3_amg (double * *r*, double * *z*, void * *data*)

block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved AMG)

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

02/19/2015

Definition at line 588 of file precondition_bcsr.c.

9.69.2.11 void fasp_precond_sweeping (double * *r*, double * *z*, void * *data*)

sweeping preconditioner for Maxwell equations

Parameters

<i>r</i>	Pointer to the vector needs preconditioning
<i>z</i>	Pointer to preconditioned vector
<i>data</i>	Pointer to precondition data

Author

Xiaozhe Hu

Date

05/01/2014

Definition at line 898 of file precondition_bcsr.c.

9.70 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.

9.70.1 Detailed Description

Preconditioners for `dBSRmat` matrices.

9.70.2 Function Documentation

9.70.2.1 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`.

9.70.2.2 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.

9.70.2.3 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.

9.70.2.4 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 precondition_bsr.c.

9.70.2.5 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 precondition_bsr.c.

9.70.2.6 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`.

9.70.2.7 `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`.

9.70.2.8 `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`.

9.70.2.9 `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`.

9.71 `precond_csr.c` File Reference

Preconditioners for [dCSRmat](#) matrices.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "forts_ns.h"
#include "mg_util.inl"
```

Functions

- `precond * fasp_precond_setup` (`SHORT` `precond_type`, `AMG_param` *`amgparam`, `ILU_param` *`iluparam`, `dCSRmat` *`A`)
Setup preconditioner interface for iterative methods.
- `void fasp_precond_diag` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
*Diagonal preconditioner $z = \text{inv}(D) * r$.*
- `void fasp_precond_ilu` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
ILU preconditioner.
- `void fasp_precond_ilu_forward` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
ILU preconditioner: only forward sweep.
- `void fasp_precond_ilu_backward` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
ILU preconditioner: only backward sweep.
- `void fasp_precond_Schwarz` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
get z from r by Schwarz
- `void fasp_precond_amg` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
AMG preconditioner.
- `void fasp_precond_famg` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
Full AMG preconditioner.
- `void fasp_precond_amli` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
AMLI AMG preconditioner.
- `void fasp_precond_nl_amli` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
Nonlinear AMLI AMG preconditioner.
- `void fasp_precond_amg_nk` (`REAL` *`r`, `REAL` *`z`, `void` *`data`)
AMG with extra near kernel solve as preconditioner.
- `void fasp_precond_free` (`SHORT` `precond_type`, `precond` *`pc`)
free preconditioner

9.71.1 Detailed Description

Preconditioners for `dCSRmat` matrices.

9.71.2 Function Documentation

9.71.2.1 `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.

9.71.2.2 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.

9.71.2.3 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.

9.71.2.4 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.

9.71.2.5 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.

9.71.2.6 void fasp_precond_free (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.

9.71.2.7 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.

9.71.2.8 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.

9.71.2.9 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.

9.71.2.10 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.

9.71.2.11 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.

9.71.2.12 precondition * fasp_precond_setup (SHORT *precond_type*, AMG_param * *amgparam*, ILU_param * *iluparam*, dCSRmat * *A*)

Setup preconditioner interface for iterative methods.

Parameters

<i>precond_type</i>	Preconditioner type
<i>amgparam</i>	Pointer to AMG parameters
<i>iluparam</i>	Pointer to ILU parameters
<i>A</i>	Pointer to the coefficient matrix

Returns

Pointer to preconditioner

Author

Feiteng Huang

Date

05/18/2009

Definition at line 32 of file precondition_csr.c.

9.72 precondition_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)

9.72.1 Detailed Description

Preconditioners for [dSTRmat](#) matrices.

9.72.2 Function Documentation

9.72.2.1 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 1707 of file precondition_str.c.

9.72.2.2 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.

9.72.2.3 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 55 of file precondition_str.c.

9.72.2.4 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 979 of file precondition_str.c.

9.72.2.5 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 816 of file precondition_str.c.

9.72.2.6 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 337 of file precondition_str.c.

9.72.2.7 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 1426 of file precondition_str.c.

9.72.2.8 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 1160 of file precondition_str.c.

9.73 pvfgmres.c File Reference

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, 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.
- **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 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, 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.

9.73.1 Detailed Description

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 [pvgmres.c](#)

9.73.2 Function Documentation

9.73.2.1 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.

9.73.2.2 `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 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 precond: 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.

9.73.2.3 `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.

9.74 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.

9.74.1 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](#)

9.74.2 Function Documentation

9.74.2.1 `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.

9.75 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.

9.75.1 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

9.75.2 Function Documentation

9.75.2.1 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.

9.75.2.2 `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.

9.75.2.3 `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.

9.75.2.4 `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.

9.76 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.

9.76.1 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.

9.76.2 Function Documentation

9.76.2.1 [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.

9.77 quadrature.c File Reference

Quadrature rules.

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

Functions

- void [fasp_quad2d](#) (INT num_qp, INT ncoor, REAL(*quad)[3])
Initialize Lagrange quadrature points and weights.
- void [fasp_gauss2d](#) (INT num_qp, INT ncoor, REAL(*gauss)[3])
Initialize Gauss quadrature points and weights.

9.77.1 Detailed Description

Quadrature rules.

9.77.2 Function Documentation

9.77.2.1 void [fasp_gauss2d](#) (INT num_qp, 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`.

9.77.2.2 `void fasp_quad2d (INT num_qp, 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`.

9.78 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 $R \cdot A \cdot P$.

9.78.1 Detailed Description

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

C-version by Ludmil Zikatanov 2010-04-08

tested 2010-04-08

9.78.2 Function Documentation

9.78.2.1 `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.

9.79 schwarz_setup.c File Reference

Setup phase for the Schwarz methods.

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

9.79.1 Detailed Description

Setup phase for the Schwarz methods.

9.79.2 Function Documentation

9.79.2.1 `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.

9.79.2.2 `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.

9.79.2.3 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.

9.79.2.4 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.

9.80 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 $n \times n$ full matrix to be a identity.

9.80.1 Detailed Description

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

9.80.2 Macro Definition Documentation

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

swap two numbers

Definition at line 9 of file smat.c.

9.80.3 Function Documentation

9.80.3.1 `INT fasp_blas_smat_inv (REAL * a, const INT n)`

Compute the inverse matrix of a small full matrix a.

Parameters

<code>a</code>	Pointer to the REAL array which stands a $n \times n$ matrix
<code>n</code>	Dimension of the matrix

Author

Xiaozhe Hu, Shiquan Zhang

Date

04/21/2010

Definition at line 554 of file smat.c.

9.80.3.2 `void fasp_blas_smat_inv_nc (REAL * a, const INT n)`

Compute the inverse of a matrix using Gauss Elimination.

Parameters

<code>a</code>	Pointer to the REAL array which stands a $n \times n$ matrix
<code>n</code>	Dimension of the matrix

Author

Xiaozhe Hu, Shiquan Zhang

Date

05/01/2010

Definition at line 405 of file smat.c.

9.80.3.3 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.

9.80.3.4 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.

9.80.3.5 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.

9.80.3.6 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.

9.80.3.7 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.

9.80.3.8 void fasp_blas_smat_invp_nc (REAL * a, const INT n)

Compute the inverse of a matrix using Gauss Elimination with Pivoting.

Parameters

a	Pointer to the REAL array which stands a $n \times n$ matrix
n	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.

9.80.3.9 REAL fasp_blas_smat_Linfinity (REAL * A, const INT n)

Compute the L infinity norm of A.

Parameters

A	Pointer to the $n \times n$ dense matrix
n	the dimension of the dense matrix

Author

Xiaozhe Hu

Date

05/26/2014

Definition at line 595 of file smat.c.

9.80.3.10 void fasp_iden_free (idenmat * A)

Free idenmat sparse matrix data memeory space.

Parameters

A	Pointer to the idenmat matrix
-----	-------------------------------

Author

Chensong Zhang

Date

2010/04/03

Definition at line 628 of file smat.c.

9.80.3.11 void fasp_smat_identity (REAL * *a*, const INT *n*, const INT *n2*)

Set a $n \times n$ full matrix to be a identity.

Parameters

a	Pointer to the REAL vector which stands for a $n \times n$ full matrix
n	Size of full matrix
$n2$	Length of the REAL vector which stores the $n \times n$ full matrix

Author

Xiaozhe Hu

Date

2010/12/25

Definition at line 728 of file smat.c.

9.80.3.12 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.

9.80.3.13 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.

9.80.3.14 void fasp_smat_identity_nc5 (REAL * a)Set a 5×5 full matrix to be a identity.

Parameters

<i>a</i>	Pointer to the REAL vector which stands for a 5*5 full matrix
----------	---

Author

Xiaozhe Hu

Date

2010/12/25

Definition at line 682 of file smat.c.

9.80.3.15 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.

9.81 smoother_bsr.c File Reference

Smoother 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.

9.81.1 Detailed Description

Smoothers for `dBSRmat` matrices.

9.81.2 Function Documentation

9.81.2.1 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 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

Author

Zhiyang Zhou

Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 08/03/2012

Definition at line 415 of file smoother_bsr.c.

9.81.2.2 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 <i>A</i>

Author

Zhiyang Zhou

Date

2010/10/25

Definition at line 535 of file smoother_bsr.c.

9.81.2.3 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 572 of file smoother_bsr.c.

9.81.2.4 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 645 of file smoother_bsr.c.

9.81.2.5 void fasp_smoother_dbsr_gs_descend (dBSRmat * A, dvector * b, dvector * u, REAL * diaginv)

Gauss-Seidel relaxation in the descending order.

Parameters

<i>A</i>	Pointer to 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 716 of file smoother_bsr.c.

9.81.2.6 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 790 of file smoother_bsr.c.

9.81.2.7 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 862 of file smoother_bsr.c.

9.81.2.8 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 940 of file smoother_bsr.c.

9.81.2.9 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 1574 of file smoother_bsr.c.

9.81.2.10 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 35 of file smoother_bsr.c.

9.81.2.11 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 259 of file smoother_bsr.c.

9.81.2.12 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 150 of file smoother_bsr.c.

9.81.2.13 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 1019 of file smoother_bsr.c.

9.81.2.14 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>diagin</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 1141 of file smoother_bsr.c.

9.81.2.15 void fasp_smoother_dbsr_sor_ascend (dBSRmat * *A*, dvector * *b*, dvector * *u*, REAL * *diagin*, 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>diagin</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 1182 of file smoother_bsr.c.

9.81.2.16 void fasp_smoother_dbsr_sor_descend (dBSRmat * *A*, dvector * *b*, dvector * *u*, REAL * *diagin*, 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 1311 of file smoother_bsr.c.

9.81.2.17 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>diaginv</i>	Inverses for all the diagonal blocks of A
<i>mark</i>	Pointer to the user-defined ordering
<i>weight</i>	Over-relaxation weight

Author

Zhiyang Zhou

Date

2010/10/25

Modified by Chunsheng Feng, Zheng Li on 2012/09/04

Definition at line 1442 of file smoother_bsr.c.

9.82 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, `INT` order, `INT` *mark, `INT` maximap, `INT` nx, `INT` ny, `INT` nz)
Colored Gauss-Seidel smoother for $Au=b$.

9.82.1 Detailed Description

Smoothers for `dCSRmat` matrices.

9.82.2 Function Documentation

- 9.82.2.1 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

<code>u</code>	Pointer to <code>dvector</code> : the unknowns (IN: initial, OUT: approximation)
<code>i_1</code>	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 09/01/2012

Definition at line 195 of file smoother_csr.c.

9.82.2.2 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.

9.82.2.3 void fasp_smoother_dcsr_gs_rb3d (dvector * *u*, dCSRmat * *A*, dvector * *b*, INT *L*, INT *order*, INT * *mark*, INT *maximap*, INT *nx*, INT *ny*, 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 1426 of file smoother_csr.c.

9.82.2.4 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.

9.82.2.5 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.

9.82.2.6 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

<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
<i>w</i>	Over-relaxation weight

Author

Xiaozhe Hu

Date

2010/11/12

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

Definition at line 1145 of file smoother_csr.c.

9.82.2.7 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.

9.82.2.8 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.

9.82.2.9 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

<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
<i>w</i>	Over-relaxation weight

Author

Xiaozhe Hu

Date

10/26/2010

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

Definition at line 745 of file smoother_csr.c.

9.82.2.10 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.

9.83 smoother_csr_cr.c File Reference

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

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

Functions

- void `fasp_smoother_dcsr_gscr` (INT *pt*, INT *n*, REAL **u*, INT **ia*, INT **ja*, REAL **a*, REAL **b*, INT *L*, INT **CF*)
Gauss Seidel method restriced to a block.

9.83.1 Detailed Description

Smoothers for `dCSRmat` matrices using compatible relaxation.

Note

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

9.83.2 Function Documentation

9.83.2.1 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`.

9.84 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

9.84.1 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.

9.84.2 Function Documentation

9.84.2.1 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`.

9.84.2.2 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.

9.85 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, [INT](#) order, [INT](#) *mark)
Gauss-Seidel method as the smoother.
- void [fasp_smoother_dstr_gs1](#) ([dSTRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [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, [INT](#) order)
Gauss method as the smoother in the C-F manner.
- void [fasp_smoother_dstr_sor](#) ([dSTRmat](#) *A, [dvector](#) *b, [dvector](#) *u, [INT](#) order, [INT](#) *mark, [REAL](#) weight)
SOR method as the smoother.

- void `fasp_smoother_dstr_sor1` (`dSTRmat *A`, `dvector *b`, `dvector *u`, `INT order`, `INT *mark`, `REAL *diaginv`, `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`, `INT order`, `REAL weight`)
SOR method as the smoother in the C-F manner.
- void `fasp_generate_diaginv_block` (`dSTRmat *A`, `ivector *neigh`, `dvector *diaginv`, `ivector *pivot`)
Generate inverse of diagonal block for block smoothers.
- void `fasp_smoother_dstr_schwarz` (`dSTRmat *A`, `dvector *b`, `dvector *u`, `dvector *diaginv`, `ivector *pivot`, `ivector *neigh`, `ivector *order`)
Schwarz method as the smoother.

9.85.1 Detailed Description

Smoothers for `dSTRmat` matrices.

9.85.2 Function Documentation

9.85.2.1 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 <code>dCSRmat</code> : the coefficient matrix
<i>neigh</i>	Pointer to <code>ivector</code> : neighborhoods
<i>diaginv</i>	Pointer to <code>dvector</code> : the inverse of the diagonals
<i>pivot</i>	Pointer to <code>ivector</code> : the pivot of diagonal blocks

Author

Xiaozhe Hu

Date

10/01/2011

Definition at line 1517 of file `smoother_str.c`.

9.85.2.2 void `fasp_smoother_dstr_gs` (`dSTRmat * A`, `dvector * b`, `dvector * u`, `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 202 of file smoother_str.c.

9.85.2.3 void fasp_smoother_dstr_gs1 (dSTRmat * *A*, dvector * *b*, dvector * *u*, 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 261 of file smoother_str.c.

9.85.2.4 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 306 of file smoother_str.c.

9.85.2.5 void fasp_smoother_dstr_gs_cf (dSTRmat * *A*, dvector * *b*, dvector * *u*, REAL * *diaginv*, INT * *mark*, 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 660 of file smoother_str.c.

9.85.2.6 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 421 of file smoother_str.c.

9.85.2.7 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 538 of file smoother_str.c.

9.85.2.8 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.

9.85.2.9 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.

9.85.2.10 void fasp_smoother_dstr_schwarz (dSTRmat * *A*, dvector * *b*, dvector * *u*, dvector * *diaginv*, ivector * *pivot*, ivector * *neigh*, ivector * *order*)

Schwarz method as the smoother.

Parameters

<i>A</i>	Pointer to 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>	Pointer to dvector: the inverse of the diagonals
<i>pivot</i>	Pointer to ivector: the pivot of diagonal blocks
<i>neigh</i>	Pointer to ivector: neighborhoods
<i>order</i>	Pointer to ivector: the smoothing order

Author

Xiaozhe Hu

Date

10/01/2011

Definition at line 1639 of file smoother_str.c.

9.85.2.11 void fasp_smoother_dstr_sor (dSTRmat * *A*, dvector * *b*, dvector * *u*, INT *order*, INT * *mark*, 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 851 of file smoother_str.c.

9.85.2.12 void fasp_smoother_dstr_sor1 (dSTRmat * *A*, dvector * *b*, dvector * *u*, INT *order*, INT * *mark*, REAL * *diaginv*, 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 912 of file smoother_str.c.

9.85.2.13 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 958 of file smoother_str.c.

9.85.2.14 void fasp_smoother_dstr_sor_cf (dSTRmat * *A*, dvector * *b*, dvector * *u*, REAL * *diaginv*, INT * *mark*, INT *order*, 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 1330 of file smoother_str.c.

9.85.2.15 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 1078 of file smoother_str.c.

9.85.2.16 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 1199 of file smoother_str.c.

9.86 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`, `INT m`, `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`, `INT m`, `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

9.86.1 Detailed Description

Sparse matrix block operations.

9.86.2 Function Documentation

9.86.2.1 void fasp_bdcsr_free (block_dCSRmat * A)

Free block CSR sparse matrix data memory space.

Parameters

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

Author

Xiaozhe Hu

Date

04/18/2014

Definition at line 30 of file `sparse_block.c`.

9.86.2.2 **SHORT** fasp_dbsr_getblk (dBSRmat * A, INT * Is, INT * Js, INT m, INT n, dBSRmat * B)

Get a sub BSR matrix of A with specified rows and columns.

Parameters

<i>A</i>	Pointer to <code>dBSRmat</code> BSR matrix
<i>B</i>	Pointer to <code>dBSRmat</code> 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 158 of file sparse_block.c.

9.86.2.3 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 254 of file sparse_block.c.

9.86.2.4 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 310 of file sparse_block.c.

9.86.2.5 **SHORT** fasp_dcsr_getblk ([dCSRmat](#) * *A*, INT * *Is*, INT * *Js*, INT *m*, 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
<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 65 of file sparse_block.c.

9.87 **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` (INT ROW, INT COL, INT NNZ, INT nb, INT storage_manner)
Create BSR sparse matrix data memory space.
- void `fasp_dbsr_alloc` (INT ROW, INT COL, INT NNZ, INT nb, 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_getdiainv` (`dBSRmat *A`)
Get D^{-1} of matrix A .
- `dBSRmat fasp_dbsr_diainv` (`dBSRmat *A`)
*Compute $B := D^{-1} * A$, where ' D ' is the block diagonal part of A .*
- `dBSRmat fasp_dbsr_diainv2` (`dBSRmat *A`, `REAL *diainv`)
*Compute $B := D^{-1} * A$, where ' D ' is the block diagonal part of A .*
- `dBSRmat fasp_dbsr_diainv3` (`dBSRmat *A`, `REAL *diainv`)
*Compute $B := D^{-1} * A$, where ' D ' is the block diagonal part of A .*
- `dBSRmat fasp_dbsr_diainv4` (`dBSRmat *A`, `REAL *diainv`)
*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 $DU = \text{diag}(U^{-1})$.*

9.87.1 Detailed Description

Sparse matrix operations for `dBSRmat` matrices.

9.87.2 Function Documentation

9.87.2.1 void fasp_dbsr_alloc (INT ROW, INT COL, INT NNZ, INT nb, 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 dBSRmat matrix

Author

Xiaozhe Hu

Date

10/26/2010

Definition at line 86 of file sparse_bsr.c.

9.87.2.2 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 180 of file sparse_bsr.c.

9.87.2.3 [dBSRmat](#) fasp_dbsr_create (INT *ROW*, INT *COL*, INT *NNZ*, INT *nb*, 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 35 of file sparse_bsr.c.

9.87.2.4 dBSRmat fasp_dbsr_diaginv (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.

9.87.2.5 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.

9.87.2.6 `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>diagin</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 763 of file sparse_bsr.c.

9.87.2.7 dBSRmat fasp_dbsr_diagin4 (dBSRmat * *A*, REAL * *diagin*)

Compute $B := D^{-1}A$, where 'D' is the block diagonal part of A.

Parameters

<i>A</i>	Pointer to the dBSRmat matrix
<i>diagin</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 1121 of file sparse_bsr.c.

9.87.2.8 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 1448 of file `sparse_bsr.c`.

9.87.2.9 `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 1676 of file `sparse_bsr.c`.

9.87.2.10 `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 291 of file sparse_bsr.c.

9.87.2.11 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 132 of file sparse_bsr.c.

9.87.2.12 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 1412 of file sparse_bsr.c.

9.87.2.13 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.

9.87.2.14 void fasp_dbsr_null (dBSRmat * A)

Initialize sparse matrix on structured grid.

Parameters

A	Pointer to the dBSRmat matrix
----------	---

Author

Xiaozhe Hu

Date

10/26/2010

Definition at line 157 of file sparse_bsr.c.

9.87.2.15 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 207 of file sparse_bsr.c.

9.88 sparse_coo.c File ReferenceSparse matrix operations for [dCOOmat](#) matrices.

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

Functions

- [dCOOmat fasp_dcoo_create](#) (INT m, INT n, 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, INT offset)
Re-index a REAL matrix in IJ format to make the index starting from 0 or 1.

9.88.1 Detailed Description

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

9.88.2 Function Documentation

9.88.2.1 void fasp_dcoo_alloc (const INT *m*, const INT *n*, const INT *nnz*, [dCOOmat](#) * *A*)

Allocate COO sparse matrix memory space.

Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros
<i>A</i>	Pointer to the dCSRmat matrix

Author

Xiaozhe Hu

Date

03/25/2013

Definition at line 62 of file sparse_coo.c.

9.88.2.2 [dCOOmat](#) fasp_dcoo_create (INT *m*, INT *n*, INT *nnz*)

Create IJ sparse matrix data memory space.

Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros

Returns

A The new [dCOOmat](#) matrix

Author

Chensong Zhang

Date

2010/04/06

Definition at line 34 of file sparse_coo.c.

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

Free IJ sparse matrix data memory space.

Parameters

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

Author

Chensong Zhang

Date

2010/04/03

Definition at line 94 of file sparse_coo.c.

9.88.2.4 void fasp_dcoo_shift ([dCOOmat](#) * *A*, *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.

9.89 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. $A_{perm}=PAP'$ by the orders given in P.*
- void `fasp_dcsr_sort` (`dCSRmat *A`)
- *Sort each row of A in ascending order w.r.t. column indices.*
- void `fasp_dcsr_getdiag` (`INT n`, `dCSRmat *A`, `dvector *diag`)
- *Get first n diagonal entries of a CSR matrix A.*
- void `fasp_dcsr_getcol` (`const INT n`, `dCSRmat *A`, `REAL *col`)
- *Get the n-th column of a CSR matrix A.*
- void `fasp_dcsr_diagpref` (`dCSRmat *A`)
- *Re-order the column and data arrays of a CSR matrix, so that the first entry in each row is the diagonal.*
- `SHORT fasp_dcsr_regdiag` (`dCSRmat *A`, `REAL value`)
- *Regularize diagonal entries of a CSR sparse matrix.*
- void `fasp_icsr_cp` (`iCSRmat *A`, `iCSRmat *B`)
- *Copy a iCSRmat to a new one B=A.*
- void `fasp_dcsr_cp` (`dCSRmat *A`, `dCSRmat *B`)
- *copy a dCSRmat to a new one B=A*
- void `fasp_icsr_trans` (`iCSRmat *A`, `iCSRmat *AT`)
- *Find transpose of iCSRmat matrix A.*
- `INT fasp_dcsr_trans` (`dCSRmat *A`, `dCSRmat *AT`)
- *Find transpose of dCSRmat matrix A.*
- void `fasp_dcsr_transpose` (`INT *row[2]`, `INT *col[2]`, `REAL *val[2]`, `INT *nn`, `INT *tniz`)
- void `fasp_dcsr_compress` (`dCSRmat *A`, `dCSRmat *B`, `REAL dtol`)
- *Compress a CSR matrix A and store in CSR matrix B by dropping small entries $abs(a_{ij}) \leq dtol$.*
- `SHORT fasp_dcsr_compress_inplace` (`dCSRmat *A`, `REAL dtol`)
- *Compress a CSR matrix A IN PLACE by dropping small entries $abs(a_{ij}) \leq dtol$.*
- void `fasp_dcsr_shift` (`dCSRmat *A`, `INT offset`)
- *Re-index a REAL matrix in CSR format to make the index starting from 0 or 1.*
- void `fasp_dcsr_symdiagscale` (`dCSRmat *A`, `dvector *diag`)
- *Symmetric diagonal scaling $D^{-1/2}AD^{-1/2}$.*
- `dCSRmat fasp_dcsr_symptat` (`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.*

9.89.1 Detailed Description

Sparse matrix operations for `dCSRmat` matrices.

9.89.2 Function Documentation

9.89.2.1 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.

9.89.2.2 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 955 of file sparse_csr.c.

9.89.2.3 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 1035 of file sparse_csr.c.

9.89.2.4 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 721 of file sparse_csr.c.

9.89.2.5 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.

9.89.2.6 `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 551 of file sparse_csr.c.

9.89.2.7 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.

9.89.2.8 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 472 of file sparse_csr.c.

9.89.2.9 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 408 of file sparse_csr.c.

9.89.2.10 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 1263 of file sparse_csr.c.

9.89.2.11 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.

9.89.2.12 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!

Modified by Chunsheng Feng, Zheng Li on 07/12/2012

Definition at line 245 of file sparse_csr.c.

9.89.2.13 SHORT fasp_dcsr_regdiag (dCSRmat * A, REAL value)

Regularize diagonal entries of a CSR sparse matrix.

Parameters

<i>A</i>	Pointer to the dCSRmat matrix
<i>value</i>	Set a value on diag(A) which is too close to zero to "value"

Returns

FASP_SUCCESS if no diagonal entry is close to zero, else ERROR

Author

Shiquan Zhang

Date

11/07/2009

Definition at line 657 of file sparse_csr.c.

9.89.2.14 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 1083 of file sparse_csr.c.

9.89.2.15 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 356 of file sparse_csr.c.

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

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

Parameters

<i>A</i>	Pointer to the dCSRmat matrix
<i>diag</i>	Pointer to the diagonal entries

Author

Xiaozhe Hu

Date

01/31/2011

Modified by Chunsheng Feng, Zheng Li on 07/11/2012

Definition at line 1144 of file sparse_csr.c.

9.89.2.17 dCSRmat fasp_dcsr_sympat (dCSRmat * A)Get symmetric part of a [dCSRmat](#) matrix.

Parameters

<i>*A</i>	pointer to the dCSRmat matrix
-----------	---

Returns

symmetrized the [dCSRmat](#) matrix

Author

Xiaozhe Hu

Date

03/21/2011

Definition at line 1230 of file sparse_csr.c.

9.89.2.18 void fasp_dcsr_trans (dCSRmat * A, dCSRmat * AT)Find transpose of [dCSRmat](#) matrix A.

Parameters

<i>A</i>	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 824 of file sparse_csr.c.

9.89.2.19 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 696 of file sparse_csr.c.

9.89.2.20 iCSRmat fasp_icsr_create (const INT *m*, const INT *n*, const INT *nnz*)

Create CSR sparse matrix data memory space.

Parameters

<i>m</i>	Number of rows
<i>n</i>	Number of columns
<i>nnz</i>	Number of nonzeros

Returns

A the new iCSRmat matrix

Author

Chensong Zhang

Date

2010/04/06

Definition at line 80 of file sparse_csr.c.

9.89.2.21 void fasp_icsr_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.

9.89.2.22 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.

9.89.2.23 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 748 of file sparse_csr.c.

9.90 sparse_csrl.c File Reference

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

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

Functions

- [dCSRLmat](#) * [fasp_dcsrl_create](#) ([INT](#) num_rows, [INT](#) num_cols, [INT](#) num_nonzeros)
Create a [dCSRLmat](#) object.
- void [fasp_dcsrl_free](#) ([dCSRLmat](#) *A)
Destroy a [dCSRLmat](#) object.

9.90.1 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.

9.90.2 Function Documentation

9.90.2.1 [dCSRLmat](#) * [fasp_dcsrl_create](#) ([INT](#) num_rows, [INT](#) num_cols, [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 29 of file sparse_csrl.c.

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

Destroy a [dCSRLmat](#) object.

Parameters

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

Author

Zhiyang Zhou

Date

01/07/2011

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

9.91 `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](#) ([INT](#) nx, [INT](#) ny, [INT](#) nz, [INT](#) nc, [INT](#) nband, [INT](#) *offsets)
Create STR sparse matrix data memory space.
- void [fasp_dstr_alloc](#) ([INT](#) nx, [INT](#) ny, [INT](#) nz, [INT](#) nxy, [INT](#) ngrid, [INT](#) nband, [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.

9.91.1 Detailed Description

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

9.91.2 Function Documentation

9.91.2.1 void [fasp_dstr_alloc](#) ([INT](#) nx, [INT](#) ny, [INT](#) nz, [INT](#) nxy, [INT](#) ngrid, [INT](#) nband, [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 107 of file sparse_str.c.

9.91.2.2 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 179 of file sparse_str.c.

9.91.2.3 [dSTRmat](#) fasp_dstr_create (INT *nx*, INT *ny*, INT *nz*, INT *nc*, 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 56 of file sparse_str.c.

9.91.2.4 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 150 of file sparse_str.c.

9.91.2.5 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.

9.92 sparse_util.c File Reference

Routines for sparse matrix operations.

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

Functions

- void `fasp_sparse_abybms_` (INT *ia, INT *ja, INT *ib, INT *jb, INT *nap, INT *map, INT *mbp, INT *ic, INT *jc)

Multiplication of two sparse matrices: calculating the nonzero structure of the result if jc is not null. If jc is null only finds num of 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 *npx, INT *icp, REAL *s)

Calculates $s = y^t x$. y is sparse, x is sparse.
- void `fasp_sparse_rapcmp_` (INT *ir, INT *jr, REAL *r, INT *ia, INT *ja, REAL *a, INT *ipt, INT *jpt, REAL *pt, INT *nin, INT *ncin, INT *iac, INT *jac, REAL *ac, INT *idummy)

*Calculates $R*A*P$ after the nonzero structure of the result is known. iac,jac,ac have to be allocated before call to this function.*
- ivector `fasp_sparse_MIS` (dCSRmat *A)

get the maximal independet set of a CSR matrix

9.92.1 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

9.92.2 Function Documentation

9.92.2.1 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 270 of file sparse_util.c.

9.92.2.2 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 122 of file sparse_util.c.

9.92.2.3 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 51 of file sparse_util.c.

9.92.2.4 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>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 357 of file sparse_util.c.

9.92.2.5 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 429 of file sparse_util.c.

9.92.2.6 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 195 of file sparse_util.c.

9.92.2.7 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 907 of file sparse_util.c.

9.92.2.8 void fasp_sparse_rapcmp_ (INT * *ir*, INT * *jr*, REAL * *r*, INT * *ia*, INT * *ja*, REAL * *a*, INT * *ipt*, INT * *jpt*, REAL * *pt*, INT * *nin*, INT * *ncin*, INT * *iac*, INT * *jac*, REAL * *ac*, INT * *idummy*)

Calculates $R \cdot A \cdot P$ after the nonzero structure of the result is known. *iac*, *jac*, *ac* have to be allocated before call to this function.

Note

:I: is input :O: is output :IO: is both

Parameters

<i>ir</i>	:I: array of row pointers for R
<i>jr</i>	:I: array of column indices for R
<i>r</i>	:I: entries of R
<i>ia</i>	:I: array of row pointers for A
<i>ja</i>	:I: array of column indices for A
<i>a</i>	:I: entries of A
<i>ipt</i>	:I: array of row pointers for P
<i>jpt</i>	:I: array of column indices for P
<i>pt</i>	:I: entries of P
<i>nin</i>	:I: number of rows in R
<i>ncin</i>	:I: number of rows in
<i>iac</i>	:O: array of row pointers for P
<i>jac</i>	:O: array of column indices for P
<i>ac</i>	:O: entries of P
<i>idummy</i>	not changed

Note

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

Definition at line 786 of file sparse_util.c.

9.92.2.9 void fasp_sparse_rapms_ (INT * *ir*, INT * *jr*, INT * *ia*, INT * *ja*, INT * *ip*, INT * *jp*, INT * *nin*, INT * *ncin*, INT * *iac*, INT * *jac*, INT * *maxrout*)

Calculates the nonzero structure of $R \cdot A \cdot P$, if *jac* is not null. If *jac* is null only finds num of nonzeros.

Note

:I: is input :O: is output :IO: is both

Parameters

<i>ir</i>	:I: array of row pointers for R
<i>jr</i>	:I: array of column indices for R
<i>ia</i>	:I: array of row pointers for A
<i>ja</i>	:I: array of column indices for A

<i>ip</i>	:I: array of row pointers for P
<i>jp</i>	:I: array of column indices for P
<i>nin</i>	:I: number of rows in R
<i>ncin</i>	:I: number of columns in R
<i>iac</i>	:O: array of row pointers for Ac
<i>jac</i>	:O: array of column indices for Ac
<i>maxrout</i>	:O: the maximum nonzeros per row for R

Note

Computes the sparsity pattern of $R \cdot A \cdot P$. maxrout is output and is the maximum nonzeros per row for r. On output we also have iac (if jac is null) and jac (if jac entry is not null). R is (nc,n) A is (n,n) and P is (n,nc)!

Modified by Chensong Zhang on 09/11/2012

Definition at line 512 of file sparse_util.c.

9.92.2.10 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 646 of file sparse_util.c.

9.92.2.11 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 593 of file sparse_util.c.

9.92.2.12 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>	:I: indices such that $y[jy]$ is nonzero
<i>y</i>	:I: is a sparse vector.
<i>nyp</i>	:I: number of nonzeros in y
<i>jx</i>	:I: indices such that $x[jx]$ is nonzero
<i>x</i>	:I: is a sparse vector.
<i>nxp</i>	:I: number of nonzeros in x
<i>icp</i>	???
<i>s</i>	:O: $s = y^t x$.

Definition at line 731 of file sparse_util.c.

9.92.2.13 void fasp_sparse_ytxbig_ (INT * *jy*, REAL * *y*, INT * *nyp*, REAL * *x*, REAL * *s*)

Calculates $s = y^t x$. *y*-sparse, *x* - no.

Note

:I: is input :O: is output :IO: is both

Parameters

<i>jy</i>	:I: indices such that $y[jy]$ is nonzero
<i>y</i>	:I: is a sparse vector.
<i>nyp</i>	:I: number of nonzeros in v
<i>x</i>	:I: also a vector assumed to have entry for any $j=jy[i]-1$; for $i=1:nyp$. This means that <i>x</i> here does not have to be sparse.
<i>s</i>	:O: $s = y^t x$.

Definition at line 697 of file sparse_util.c.

9.93 spbcgs.c File Reference

Krylov subspace methods – Preconditioned BiCGstab with safe net.

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

Functions

- **INT fasp_solver_dcsr_spgs** (dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)
Preconditioned BiCGstab method for solving $Au=b$ with safe net.
- **INT fasp_solver_dbsr_spgs** (dBSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)
Preconditioned BiCGstab method for solving $Au=b$ with safe net.
- **INT fasp_solver_bdcsr_spgs** (block_dCSRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)
Preconditioned BiCGstab method for solving $Au=b$ with safe net.
- **INT fasp_solver_dstsr_spgs** (dSTRmat *A, dvector *b, dvector *u, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtIvl)
Preconditioned BiCGstab method for solving $Au=b$ with safe net.

9.93.1 Detailed Description

Krylov subspace methods – Preconditioned BiCGstab with safe 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

Safe 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

9.93.2 Function Documentation

9.93.2.1 **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 safe 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 871 of file spbcgs.c.

9.93.2.2 `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 safe 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 481 of file spbcgs.c.

9.93.2.3 `INT fasp_solver_dcsr_spbcgs (dCSRmat * A, dvector * b, dvector * u, precondition * pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl)`

Preconditioned BiCGstab method for solving $Au=b$ with safe 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 91 of file spbcgs.c.

9.93.2.4 `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 safe 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 1261 of file spbcgs.c.

9.94 spcg.c File Reference

Krylov subspace methods – Preconditioned conjugate gradient with safe 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, precondition *pc, const REAL tol, const INT MaxIt, const SHORT stop_type, const SHORT prtlvl)`
Preconditioned conjugate gradient method for solving $Au=b$ with safe net.
- `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 safe net.
- `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 safe net.

9.94.1 Detailed Description

Krylov subspace methods – Preconditioned conjugate gradient with safe net.

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: $x_{k+1} = x_k + \alpha p_k$;
- check whether x is NAN;
- perform stagnation check;
- update residual: $r_{k+1} = r_k - \alpha(Ap_k)$;
- 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

Safe 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 [pcg.c](#) for a version without safe net

9.94.2 Function Documentation

9.94.2.1 **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 safe 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.

9.94.2.2 `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 safe 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.

9.94.2.3 `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 safe 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.

9.95 spgmres.c File Reference

Krylov subspace methods – Preconditioned GMRes with safe 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.

9.95.1 Detailed Description

Krylov subspace methods – Preconditioned GMRes with safe 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 safe net

9.95.2 Function Documentation

9.95.2.1 **INT 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 *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.

9.95.2.2 **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.

9.95.2.3 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.

9.95.2.4 **INT** fasp_solver_dstr_spgmres (**dSTRmat** * *A*, **dvector** * *b*, **dvector** * *x*, **precond** * *pc*, const **REAL** *tol*, const **INT** *MaxIt*, **SHORT** *restart*, const **SHORT** *stop_type*, const **SHORT** *prtlvl*)

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.

9.96 spminres.c File Reference

Krylov subspace methods – Preconditioned minimal residual with safe 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*, **precond** **pc*, const **REAL** *tol*, const **INT** *MaxIt*, const **SHORT** *stop_type*, const **SHORT** *prtlvl*)
A preconditioned minimal residual (Minres) method for solving $Au=b$ with safe net.
- **INT** fasp_solver_bdcsr_spminres (**block_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 $Au=b$ with safe net.
- **INT** fasp_solver_dstr_spminres (**dSTRmat** **A*, **dvector** **b*, **dvector** **u*, **precond** **pc*, const **REAL** *tol*, const **INT** *MaxIt*, const **SHORT** *stop_type*, const **SHORT** *prtlvl*)
A preconditioned minimal residual (Minres) method for solving $Au=b$ with safe net.

9.96.1 Detailed Description

Krylov subspace methods – Preconditioned minimal residual with safe net.

Abstract algorithm

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: $x_{k+1} = x_k + \alpha*p_k$;
- 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

Safe 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 safe net

9.96.2 Function Documentation

9.96.2.1 **INT fasp_solver_bdcsr_spmminres (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 safe 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

04/09/2013

Definition at line 544 of file spminres.c.

9.96.2.2 **INT fasp_solver_dcsr_spmminres (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 safe 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.

9.96.2.3 `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 safe 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 spminres.c.

9.97 spvgmres.c File Reference

Krylov subspace methods – Preconditioned variable-restart GMRes with safe 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](#) prtvl)
- 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](#) prtvl)
- 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](#) prtvl)
- 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](#) prtvl)
- Solve "Ax=b" using PGMRES(right preconditioned) iterative method in which the restart parameter can be adaptively modified during the iteration.*

9.97.1 Detailed Description

Krylov subspace methods – Preconditioned variable-restart GMRes with safe 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 safe net

9.97.2 Function Documentation

9.97.2.1 [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](#) prtvl)

Preconditioned GMRES method for solving Au=b.

Parameters

A	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.

9.97.2.2 `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.

9.97.2.3 **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*)

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.

9.97.2.4 `INT 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 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.

9.98 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

9.98.1 Detailed Description

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

9.98.2 Function Documentation

9.98.2.1 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.

9.98.2.2 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.

9.98.3 Variable Documentation**9.98.3.1 INT THDs_AMG_GS =0**

AMG GS smoothing threads

Definition at line 107 of file threads.c.

9.98.3.2 INT THDs_CPR_gGS =0

global matrix GS smoothing threads

Definition at line 109 of file threads.c.

9.98.3.3 INT THDs_CPR_IGS =0

reservoir GS smoothing threads

Definition at line 108 of file threads.c.

9.99 timing.c File Reference

Timing subroutines.

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

Functions

- void [fasp_gettime](#) (REAL *time)

Get system time.

9.99.1 Detailed Description

Timing subroutines.

9.99.2 Function Documentation

9.99.2.1 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.

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

9.100.1 Detailed Description

Simple operations for vectors.

Note

Every structures should be initialized before usage.

9.100.2 Function Documentation

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

Create dvector data space of REAL type.

Parameters

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

Author

Chensong Zhang

Date

2010/04/06

Definition at line 99 of file `vec.c`.

9.100.2.2 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.

9.100.2.3 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.

9.100.2.4 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.

9.100.2.5 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.

9.100.2.6 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.

9.100.2.7 void fasp_dvec_null (dvector * x)

Initialize dvector.

Parameters

x	Pointer to dvector which needs to be initialized
-----	--

Author

Chensong Zhang

Date

2010/04/03

Definition at line 177 of file vec.c.

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

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

Parameters

n	Size of the vector
x	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.

9.100.2.9 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.

9.100.2.10 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.

9.100.2.11 void fasp_ivec_alloc (const INT *m*, ivec * *u*)

Create vector data space of INT type.

Parameters

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

Author

Chensong Zhang

Date

2010/04/06

Definition at line 119 of file vec.c.

9.100.2.12 `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.

9.100.2.13 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.

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

Set ivector value to be m .

Parameters

m	Integer value of ivector
u	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.

9.101 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)

9.101.1 Detailed Description

Wrappers for accessing functions by advanced users.

TODO: Input variables should not need [fasp.h](#)!!! –Chensong

9.101.2 Function Documentation

9.101.2.1 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 37 of file wrapper.c.

9.101.2.2 void fasp_fwapper_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 87 of file wrapper.c.

9.101.2.3 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 146 of file wrapper.c.

9.101.2.4 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 232 of file wrapper.c.

Index

__FASP_HEADER__
fasp.h, [154](#)

A

precond_FASP_blkoi_data, [61](#)
precond_sweeping_data, [65](#)

A_diag

precond_block_data, [50](#)

ABS

fasp.h, [154](#)

AMG_ILU_levels

input_param, [40](#)

AMG_Schwarz_levels

input_param, [41](#)

AMG_aggregation_type

input_param, [39](#)

AMG_aggressive_level

input_param, [39](#)

AMG_aggressive_path

input_param, [39](#)

AMG_amli_degree

input_param, [39](#)

AMG_coarse_dof

input_param, [39](#)

AMG_coarse_scaling

input_param, [40](#)

AMG_coarse_solver

input_param, [40](#)

AMG_coarsening_type

input_param, [40](#)

AMG_cycle_type

input_param, [40](#)

AMG_data, [19](#)

AMG_data_bsr, [20](#)

AMG_interpolation_type

input_param, [40](#)

AMG_levels

input_param, [40](#)

AMG_max_aggregation

input_param, [40](#)

AMG_max_row_sum

input_param, [40](#)

AMG_maxit

input_param, [41](#)

AMG_nl_amli_krylov_type

input_param, [41](#)

AMG_pair_number

input_param, [41](#)

AMG_param, [22](#)

AMG_polynomial_degree

input_param, [41](#)

AMG_postsmooth_iter

input_param, [41](#)

AMG_presmooth_iter

input_param, [41](#)

AMG_quality_bound

input_param, [41](#)

AMG_relaxation

input_param, [41](#)

AMG_smooth_filter

input_param, [42](#)

AMG_smooth_order

input_param, [42](#)

AMG_smoother

input_param, [42](#)

AMG_strong_coupled

input_param, [42](#)

AMG_strong_threshold

input_param, [42](#)

AMG_tentative_smooth

input_param, [42](#)

AMG_tol

input_param, [42](#)

AMG_truncation_threshold

input_param, [42](#)

AMG_type

input_param, [42](#)

AMLI_CYCLE

fasp_const.h, [165](#)

ASCEND

fasp_const.h, [165](#)

Abcsr

precond_block_data, [50](#)

Ai

precond_sweeping_data, [65](#)

amg.c, [69](#)

fasp_solver_amg, [69](#)

amg_setup_cr.c, [70](#)

fasp_amg_setup_cr, [70](#)

amg_setup_rs.c, [71](#)

fasp_amg_setup_rs, [71](#)

- amg_setup_sa.c, [72](#)
 - fasp_amg_setup_sa, [73](#)
 - fasp_amg_setup_sa_bsr, [73](#)
- amg_setup_ua.c, [74](#)
 - fasp_amg_setup_ua, [74](#)
 - fasp_amg_setup_ua_bsr, [74](#)
- amg_solve.c, [76](#)
 - fasp_amg_solve, [77](#)
 - fasp_amg_solve_amli, [78](#)
 - fasp_amg_solve_nl_amli, [78](#)
 - fasp_famg_solve, [79](#)
- amgparam
 - precond_block_data, [50](#)
- amlirecur.c, [79](#)
 - fasp_amg_amli_coef, [80](#)
 - fasp_solver_amli, [80](#)
 - fasp_solver_nl_amli, [81](#)
 - fasp_solver_nl_amli_bsr, [81](#)
- array.c, [82](#)
 - fasp_array_cp, [83](#)
 - fasp_array_cp_nc3, [83](#)
 - fasp_array_cp_nc5, [83](#)
 - fasp_array_cp_nc7, [84](#)
 - fasp_array_null, [84](#)
 - fasp_array_set, [85](#)
 - fasp_iarray_cp, [85](#)
 - fasp_iarray_set, [85](#)
- BIGREAL
 - fasp_const.h, [165](#)
- blas_array.c, [87](#)
 - fasp_blas_array_ax, [88](#)
 - fasp_blas_array_axpby, [88](#)
 - fasp_blas_array_axpy, [89](#)
 - fasp_blas_array_axpyz, [89](#)
 - fasp_blas_array_dotprod, [89](#)
 - fasp_blas_array_norm1, [91](#)
 - fasp_blas_array_norm2, [91](#)
 - fasp_blas_array_norminf, [92](#)
- blas_bcsr.c, [92](#)
 - fasp_blas_bdbsr_aAxy, [93](#)
 - fasp_blas_bdbsr_mxv, [93](#)
 - fasp_blas_bdcsr_aAxy, [94](#)
 - fasp_blas_bdcsr_mxv, [94](#)
- blas_bsr.c, [95](#)
 - fasp_blas_dbsr_aAxy, [95](#)
 - fasp_blas_dbsr_aAxy, [96](#)
 - fasp_blas_dbsr_aAxy_agg, [96](#)
 - fasp_blas_dbsr_axm, [98](#)
 - fasp_blas_dbsr_mxm, [98](#)
 - fasp_blas_dbsr_mxv, [99](#)
 - fasp_blas_dbsr_mxv_agg, [99](#)
 - fasp_blas_dbsr_rap, [100](#)
 - fasp_blas_dbsr_rap1, [100](#)
 - fasp_blas_dbsr_rap_agg, [101](#)
- blas_csr.c, [101](#)
 - fasp_blas_dcsr_aAxy, [103](#)
 - fasp_blas_dcsr_aAxy_agg, [103](#)
 - fasp_blas_dcsr_add, [103](#)
 - fasp_blas_dcsr_axm, [104](#)
 - fasp_blas_dcsr_bandwidth, [104](#)
 - fasp_blas_dcsr_mxm, [105](#)
 - fasp_blas_dcsr_mxv, [105](#)
 - fasp_blas_dcsr_mxv_agg, [106](#)
 - fasp_blas_dcsr_ptap, [106](#)
 - fasp_blas_dcsr_rap, [107](#)
 - fasp_blas_dcsr_rap4, [107](#)
 - fasp_blas_dcsr_rap_agg, [108](#)
 - fasp_blas_dcsr_rap_agg1, [108](#)
 - fasp_blas_dcsr_vmv, [109](#)
- blas_csrl.c, [109](#)
 - fasp_blas_dcsrl_mxv, [110](#)
- blas_smat.c, [110](#)
 - fasp_blas_array_axpy_nc2, [112](#)
 - fasp_blas_array_axpy_nc3, [112](#)
 - fasp_blas_array_axpy_nc5, [113](#)
 - fasp_blas_array_axpy_nc7, [113](#)
 - fasp_blas_array_axpyz_nc2, [113](#)
 - fasp_blas_array_axpyz_nc3, [114](#)
 - fasp_blas_array_axpyz_nc5, [114](#)
 - fasp_blas_array_axpyz_nc7, [115](#)
 - fasp_blas_smat_aAxy, [115](#)
 - fasp_blas_smat_add, [116](#)
 - fasp_blas_smat_axm, [116](#)
 - fasp_blas_smat_mul, [117](#)
 - fasp_blas_smat_mul_nc2, [117](#)
 - fasp_blas_smat_mul_nc3, [117](#)
 - fasp_blas_smat_mul_nc5, [119](#)
 - fasp_blas_smat_mul_nc7, [119](#)
 - fasp_blas_smat_mxv, [119](#)
 - fasp_blas_smat_mxv_nc2, [121](#)
 - fasp_blas_smat_mxv_nc3, [121](#)
 - fasp_blas_smat_mxv_nc5, [121](#)
 - fasp_blas_smat_mxv_nc7, [123](#)
 - fasp_blas_smat_ymAx, [123](#)
 - fasp_blas_smat_ymAx_nc2, [123](#)
 - fasp_blas_smat_ymAx_nc3, [125](#)
 - fasp_blas_smat_ymAx_nc5, [125](#)
 - fasp_blas_smat_ymAx_nc7, [126](#)
 - fasp_blas_smat_ymAx_ns, [126](#)
 - fasp_blas_smat_ymAx_ns2, [127](#)
 - fasp_blas_smat_ymAx_ns3, [127](#)
 - fasp_blas_smat_ymAx_ns5, [128](#)
 - fasp_blas_smat_ymAx_ns7, [128](#)
 - fasp_blas_smat_ypAx, [129](#)
 - fasp_blas_smat_ypAx_nc2, [129](#)
 - fasp_blas_smat_ypAx_nc3, [130](#)
 - fasp_blas_smat_ypAx_nc5, [130](#)

- fasp_blas_smat_ypAx_nc7, 130
- blas_str.c, 131
 - fasp_blas_dstr_aAxy, 131
 - fasp_blas_dstr_mxv, 132
 - fasp_dstr_diagscale, 132
- blas_vec.c, 133
 - fasp_blas_dvec_axpy, 133
 - fasp_blas_dvec_axpyz, 134
 - fasp_blas_dvec_dotprod, 134
 - fasp_blas_dvec_norm1, 135
 - fasp_blas_dvec_norm2, 135
 - fasp_blas_dvec_norminf, 136
 - fasp_blas_dvec_reterr, 136
- block_BSR, 24
 - fasp_block.h, 160
- block_Reservoir, 27
 - fasp_block.h, 161
- block_dCSRmat, 24
 - fasp_block.h, 160
- block_dvector, 25
 - fasp_block.h, 160
- block_iCSRmat, 25
 - fasp_block.h, 161
- block_ivector, 26
 - fasp_block.h, 161
- CF_ORDER
 - fasp_const.h, 165
- CGPT
 - fasp_const.h, 165
- CLASSIC_AMG
 - fasp_const.h, 165
- COARSE_AC
 - fasp_const.h, 165
- COARSE_CR
 - fasp_const.h, 166
- COARSE_MIS
 - fasp_const.h, 166
- COARSE_RS
 - fasp_const.h, 166
- COARSE_RSP
 - fasp_const.h, 166
- CPFIRST
 - fasp_const.h, 166
- checkmat.c, 137
 - fasp_check_dCSRmat, 138
 - fasp_check_diagdom, 138
 - fasp_check_diagpos, 138
 - fasp_check_diagzero, 140
 - fasp_check_iCSRmat, 140
 - fasp_check_symm, 140
- coarsening_cr.c, 142
 - fasp_amg_coarsening_cr, 142
- coarsening_rs.c, 143
 - fasp_amg_coarsening_rs, 144
- convert.c, 145
 - endian_convert_int, 146
 - endian_convert_real, 146
 - fasp_aux_bbyteToldouble, 146
 - fasp_aux_change_endian4, 148
 - fasp_aux_change_endian8, 148
- count
 - fasp.h, 158
- dBSRmat, 27
 - fasp_block.h, 161
 - JA, 28
 - val, 28
- dCOOmat, 28
 - fasp.h, 157
- dCSRmat, 29
 - fasp.h, 157
- dCSRmat, 30
 - fasp.h, 157
- dCSRmat2SAMGInput
 - interface_samg.c, 215
- DESCEND
 - fasp_const.h, 166
- DIAGONAL_PREF
 - fasp.h, 154
- DLMALLOC
 - fasp.h, 154
- dSTRmat, 31
 - fasp.h, 157
- ddeinmat, 30
 - fasp.h, 157
- diag
 - precond_block_reservoir_data, 52
- diaginv
 - precond_FASP_blkoi_data, 61
 - precond_block_reservoir_data, 52
- diaginv_S
 - precond_FASP_blkoi_data, 62
- diaginv_noscale
 - precond_FASP_blkoi_data, 61
- diaginvS
 - precond_block_reservoir_data, 52
- dlength
 - io.c, 243
- doxygen.h, 149
- dvector, 32
 - fasp.h, 157
- dvector2SAMGInput
 - interface_samg.c, 215
- e
 - grid2d, 33
- ERROR_ALLOC_MEM
 - fasp_const.h, 166

ERROR_AMG_COARSE_TYPE
 fasp_const.h, 166
 ERROR_AMG_COARSEING
 fasp_const.h, 167
 ERROR_AMG_INTERP_TYPE
 fasp_const.h, 167
 ERROR_AMG_SMOOTH_TYPE
 fasp_const.h, 167
 ERROR_DATA_STRUCTURE
 fasp_const.h, 167
 ERROR_DATA_ZERODIAG
 fasp_const.h, 167
 ERROR_DUMMY_VAR
 fasp_const.h, 167
 ERROR_INPUT_PAR
 fasp_const.h, 167
 ERROR_LIC_TYPE
 fasp_const.h, 167
 ERROR_MAT_SIZE
 fasp_const.h, 167
 ERROR_MISC
 fasp_const.h, 168
 ERROR_NUM_BLOCKS
 fasp_const.h, 168
 ERROR_OPEN_FILE
 fasp_const.h, 168
 ERROR_QUAD_DIM
 fasp_const.h, 168
 ERROR_QUAD_TYPE
 fasp_const.h, 168
 ERROR_REGRESS
 fasp_const.h, 168
 ERROR_SOLVER_EXIT
 fasp_const.h, 168
 ERROR_SOLVER_ILUSETUP
 fasp_const.h, 168
 ERROR_SOLVER_MAXIT
 fasp_const.h, 168
 ERROR_SOLVER_MISC
 fasp_const.h, 169
 ERROR_SOLVER_PRECTYPE
 fasp_const.h, 169
 ERROR_SOLVER_SOLSTAG
 fasp_const.h, 169
 ERROR_SOLVER_STAG
 fasp_const.h, 169
 ERROR_SOLVER_TOLSMALL
 fasp_const.h, 169
 ERROR_SOLVER_TYPE
 fasp_const.h, 169
 ERROR_UNKNOWN
 fasp_const.h, 169
 ERROR_WRONG_FILE
 fasp_const.h, 169
 edges
 grid2d, 33
 ediri
 grid2d, 33
 efather
 grid2d, 33
 eigen.c, 149
 fasp_dcsr_eig, 149
 endian_convert_int
 convert.c, 146
 endian_convert_real
 convert.c, 146
 FALSE
 fasp_const.h, 169
 FASP_GET_START_END
 threads.c, 448
 FASP_GSRB
 fasp.h, 154
 FASP_SUCCESS
 fasp_const.h, 170
 FASP_USE_ILU
 fasp.h, 155
 FGPT
 fasp_const.h, 170
 FPFIRST
 fasp_const.h, 170
 factor.f, 150
 famg.c, 150
 fasp_solver_famg, 151
 fasp.h, 151
 __FASP_HEADER__, 154
 ABS, 154
 count, 158
 dCOOmat, 157
 dCSRLmat, 157
 dCSRmat, 157
 DIAGONAL_PREF, 154
 DLMALLOC, 154
 dSTRmat, 157
 ddenmat, 157
 dvector, 157
 FASP_GSRB, 154
 FASP_USE_ILU, 155
 GE, 155
 GT, 155
 grid2d, 157
 iCOOmat, 157
 iCSRmat, 157
 IMAP, 158
 INT, 155
 ISNAN, 155
 idenmat, 158
 ivector, 158

- LE, [155](#)
- LONG, [155](#)
- LONGLONG, [155](#)
- LS, [156](#)
- LinkedList, [158](#)
- ListElement, [158](#)
- MAX, [156](#)
- MAXIMAP, [158](#)
- MIN, [156](#)
- NEDMALLOC, [156](#)
- nx_rb, [158](#)
- ny_rb, [159](#)
- nz_rb, [159](#)
- PUT_INT, [156](#)
- PUT_REAL, [156](#)
- pcgrid2d, [158](#)
- pgrid2d, [158](#)
- REAL, [156](#)
- RS_C1, [156](#)
- SHORT, [157](#)
- total_alloc_count, [159](#)
- total_alloc_mem, [159](#)
- fasp_BinarySearch
 - ordering.c, [277](#)
- fasp_Schwarz_data_free
 - init.c, [210](#)
- fasp_Schwarz_get_block_matrix
 - schwarz_setup.c, [355](#)
- fasp_Schwarz_setup
 - schwarz_setup.c, [355](#)
- fasp_amg_amli_coef
 - amlirecur.c, [80](#)
- fasp_amg_coarsening_cr
 - coarsening_cr.c, [142](#)
- fasp_amg_coarsening_rs
 - coarsening_rs.c, [144](#)
- fasp_amg_data_bsr_create
 - init.c, [205](#)
- fasp_amg_data_bsr_free
 - init.c, [206](#)
- fasp_amg_data_create
 - init.c, [206](#)
- fasp_amg_data_free
 - init.c, [206](#)
- fasp_amg_interp
 - interpolation.c, [218](#)
- fasp_amg_interp1
 - interpolation.c, [218](#)
- fasp_amg_interp_em
 - interpolation_em.c, [220](#)
- fasp_amg_interp_trunc
 - interpolation.c, [219](#)
- fasp_amg_setup_cr
 - amg_setup_cr.c, [70](#)
- fasp_amg_setup_rs
 - amg_setup_rs.c, [71](#)
- fasp_amg_setup_sa
 - amg_setup_sa.c, [73](#)
- fasp_amg_setup_sa_bsr
 - amg_setup_sa.c, [73](#)
- fasp_amg_setup_ua
 - amg_setup_ua.c, [74](#)
- fasp_amg_setup_ua_bsr
 - amg_setup_ua.c, [74](#)
- fasp_amg_solve
 - amg_solve.c, [77](#)
- fasp_amg_solve_amli
 - amg_solve.c, [78](#)
- fasp_amg_solve_nl_amli
 - amg_solve.c, [78](#)
- fasp_array_cp
 - array.c, [83](#)
- fasp_array_cp_nc3
 - array.c, [83](#)
- fasp_array_cp_nc5
 - array.c, [83](#)
- fasp_array_cp_nc7
 - array.c, [84](#)
- fasp_array_null
 - array.c, [84](#)
- fasp_array_set
 - array.c, [85](#)
- fasp_aux_bbyteToldouble
 - convert.c, [146](#)
- fasp_aux_change_endian4
 - convert.c, [148](#)
- fasp_aux_change_endian8
 - convert.c, [148](#)
- fasp_aux_dQuickSort
 - ordering.c, [274](#)
- fasp_aux_dQuickSortIndex
 - ordering.c, [274](#)
- fasp_aux_givens
 - givens.c, [188](#)
- fasp_aux_iQuickSort
 - ordering.c, [275](#)
- fasp_aux_iQuickSortIndex
 - ordering.c, [275](#)
- fasp_aux_merge
 - ordering.c, [276](#)
- fasp_aux_msort
 - ordering.c, [276](#)
- fasp_aux_unique
 - ordering.c, [277](#)
- fasp_bdcsr_free
 - sparse_block.c, [391](#)
- fasp_blas_array_ax
 - blas_array.c, [88](#)

fasp_blas_array_axpby
 blas_array.c, [88](#)
 fasp_blas_array_axpy
 blas_array.c, [89](#)
 fasp_blas_array_axpy_nc2
 blas_smat.c, [112](#)
 fasp_blas_array_axpy_nc3
 blas_smat.c, [112](#)
 fasp_blas_array_axpy_nc5
 blas_smat.c, [113](#)
 fasp_blas_array_axpy_nc7
 blas_smat.c, [113](#)
 fasp_blas_array_axpyz
 blas_array.c, [89](#)
 fasp_blas_array_axpyz_nc2
 blas_smat.c, [113](#)
 fasp_blas_array_axpyz_nc3
 blas_smat.c, [114](#)
 fasp_blas_array_axpyz_nc5
 blas_smat.c, [114](#)
 fasp_blas_array_axpyz_nc7
 blas_smat.c, [115](#)
 fasp_blas_array_dotprod
 blas_array.c, [89](#)
 fasp_blas_array_norm1
 blas_array.c, [91](#)
 fasp_blas_array_norm2
 blas_array.c, [91](#)
 fasp_blas_array_norminf
 blas_array.c, [92](#)
 fasp_blas_bdbsr_aAxy
 blas_bcsr.c, [93](#)
 fasp_blas_bdbsr_mxv
 blas_bcsr.c, [93](#)
 fasp_blas_bdcsr_aAxy
 blas_bcsr.c, [94](#)
 fasp_blas_bdcsr_mxv
 blas_bcsr.c, [94](#)
 fasp_blas_dbsr_aAxpby
 blas_bsr.c, [95](#)
 fasp_blas_dbsr_aAxy
 blas_bsr.c, [96](#)
 fasp_blas_dbsr_aAxy_agg
 blas_bsr.c, [96](#)
 fasp_blas_dbsr_axm
 blas_bsr.c, [98](#)
 fasp_blas_dbsr_mxm
 blas_bsr.c, [98](#)
 fasp_blas_dbsr_mxv
 blas_bsr.c, [99](#)
 fasp_blas_dbsr_mxv_agg
 blas_bsr.c, [99](#)
 fasp_blas_dbsr_rap
 blas_bsr.c, [100](#)
 fasp_blas_dbsr_rap1
 blas_bsr.c, [100](#)
 fasp_blas_dbsr_rap_agg
 blas_bsr.c, [101](#)
 fasp_blas_dcsr_aAxy
 blas_csr.c, [103](#)
 fasp_blas_dcsr_aAxy_agg
 blas_csr.c, [103](#)
 fasp_blas_dcsr_add
 blas_csr.c, [103](#)
 fasp_blas_dcsr_axm
 blas_csr.c, [104](#)
 fasp_blas_dcsr_bandwidth
 blas_csr.c, [104](#)
 fasp_blas_dcsr_mxm
 blas_csr.c, [105](#)
 fasp_blas_dcsr_mxv
 blas_csr.c, [105](#)
 fasp_blas_dcsr_mxv_agg
 blas_csr.c, [106](#)
 fasp_blas_dcsr_ptap
 blas_csr.c, [106](#)
 fasp_blas_dcsr_rap
 blas_csr.c, [107](#)
 fasp_blas_dcsr_rap2
 rap.c, [353](#)
 fasp_blas_dcsr_rap4
 blas_csr.c, [107](#)
 fasp_blas_dcsr_rap_agg
 blas_csr.c, [108](#)
 fasp_blas_dcsr_rap_agg1
 blas_csr.c, [108](#)
 fasp_blas_dcsr_vmv
 blas_csr.c, [109](#)
 fasp_blas_dcsr_l_mxv
 blas_csr.c, [110](#)
 fasp_blas_dstr_aAxy
 blas_str.c, [131](#)
 fasp_blas_dstr_mxv
 blas_str.c, [132](#)
 fasp_blas_dvec_axpy
 blas_vec.c, [133](#)
 fasp_blas_dvec_axpyz
 blas_vec.c, [134](#)
 fasp_blas_dvec_dotprod
 blas_vec.c, [134](#)
 fasp_blas_dvec_norm1
 blas_vec.c, [135](#)
 fasp_blas_dvec_norm2
 blas_vec.c, [135](#)
 fasp_blas_dvec_norminf
 blas_vec.c, [136](#)
 fasp_blas_dvec_relerr
 blas_vec.c, [136](#)

- fasp_blas_smat_Linfinity
smat.c, [360](#)
- fasp_blas_smat_aAxpby
blas_smat.c, [115](#)
- fasp_blas_smat_add
blas_smat.c, [116](#)
- fasp_blas_smat_axm
blas_smat.c, [116](#)
- fasp_blas_smat_inv
smat.c, [357](#)
- fasp_blas_smat_inv_nc
smat.c, [357](#)
- fasp_blas_smat_inv_nc2
smat.c, [358](#)
- fasp_blas_smat_inv_nc3
smat.c, [358](#)
- fasp_blas_smat_inv_nc4
smat.c, [358](#)
- fasp_blas_smat_inv_nc5
smat.c, [359](#)
- fasp_blas_smat_inv_nc7
smat.c, [359](#)
- fasp_blas_smat_invp_nc
smat.c, [359](#)
- fasp_blas_smat_mul
blas_smat.c, [117](#)
- fasp_blas_smat_mul_nc2
blas_smat.c, [117](#)
- fasp_blas_smat_mul_nc3
blas_smat.c, [117](#)
- fasp_blas_smat_mul_nc5
blas_smat.c, [119](#)
- fasp_blas_smat_mul_nc7
blas_smat.c, [119](#)
- fasp_blas_smat_m xv
blas_smat.c, [119](#)
- fasp_blas_smat_m xv_nc2
blas_smat.c, [121](#)
- fasp_blas_smat_m xv_nc3
blas_smat.c, [121](#)
- fasp_blas_smat_m xv_nc5
blas_smat.c, [121](#)
- fasp_blas_smat_m xv_nc7
blas_smat.c, [123](#)
- fasp_blas_smat_ymAx
blas_smat.c, [123](#)
- fasp_blas_smat_ymAx_nc2
blas_smat.c, [123](#)
- fasp_blas_smat_ymAx_nc3
blas_smat.c, [125](#)
- fasp_blas_smat_ymAx_nc5
blas_smat.c, [125](#)
- fasp_blas_smat_ymAx_nc7
blas_smat.c, [126](#)
- fasp_blas_smat_ymAx_ns
blas_smat.c, [126](#)
- fasp_blas_smat_ymAx_ns2
blas_smat.c, [127](#)
- fasp_blas_smat_ymAx_ns3
blas_smat.c, [127](#)
- fasp_blas_smat_ymAx_ns5
blas_smat.c, [128](#)
- fasp_blas_smat_ymAx_ns7
blas_smat.c, [128](#)
- fasp_blas_smat_ypAx
blas_smat.c, [129](#)
- fasp_blas_smat_ypAx_nc2
blas_smat.c, [129](#)
- fasp_blas_smat_ypAx_nc3
blas_smat.c, [130](#)
- fasp_blas_smat_ypAx_nc5
blas_smat.c, [130](#)
- fasp_blas_smat_ypAx_nc7
blas_smat.c, [130](#)
- fasp_block.h, [159](#)
 - block_BSR, [160](#)
 - block_Reservoir, [161](#)
 - block_dCSRmat, [160](#)
 - block_dvector, [160](#)
 - block_iCSRmat, [161](#)
 - block_ivector, [161](#)
 - dBSRmat, [161](#)
 - precond_block_reservoir_data, [161](#)
- fasp_check_dCSRmat
checkmat.c, [138](#)
- fasp_check_diagdom
checkmat.c, [138](#)
- fasp_check_diagpos
checkmat.c, [138](#)
- fasp_check_diagzero
checkmat.c, [140](#)
- fasp_check_iCSRmat
checkmat.c, [140](#)
- fasp_check_symm
checkmat.c, [140](#)
- fasp_chkerr
message.c, [268](#)
- fasp_const.h, [161](#)
 - AMLI_CYCLE, [165](#)
 - ASCEND, [165](#)
 - BIGREAL, [165](#)
 - CF_ORDER, [165](#)
 - CGPT, [165](#)
 - CLASSIC_AMG, [165](#)
 - COARSE_AC, [165](#)
 - COARSE_CR, [166](#)
 - COARSE_MIS, [166](#)
 - COARSE_RS, [166](#)

COARSE_RSP, 166
CPFIRST, 166
DESCEND, 166
ERROR_ALLOC_MEM, 166
ERROR_AMG_COARSE_TYPE, 166
ERROR_AMG_COARSEING, 167
ERROR_AMG_INTERP_TYPE, 167
ERROR_AMG_SMOOTH_TYPE, 167
ERROR_DATA_STRUCTURE, 167
ERROR_DATA_ZERODIAG, 167
ERROR_DUMMY_VAR, 167
ERROR_INPUT_PAR, 167
ERROR_LIC_TYPE, 167
ERROR_MAT_SIZE, 167
ERROR_MISC, 168
ERROR_NUM_BLOCKS, 168
ERROR_OPEN_FILE, 168
ERROR_QUAD_DIM, 168
ERROR_QUAD_TYPE, 168
ERROR_REGRESS, 168
ERROR_SOLVER_EXIT, 168
ERROR_SOLVER_ILUSETUP, 168
ERROR_SOLVER_MAXIT, 168
ERROR_SOLVER_MISC, 169
ERROR_SOLVER_PRECTYPE, 169
ERROR_SOLVER_SOLSTAG, 169
ERROR_SOLVER_STAG, 169
ERROR_SOLVER_TOLSMALL, 169
ERROR_SOLVER_TYPE, 169
ERROR_UNKNOWN, 169
ERROR_WRONG_FILE, 169
FALSE, 169
FASP_SUCCESS, 170
FGPT, 170
FPFIRST, 170
G0PT, 170
ILUk, 170
ILUt, 170
ILUtp, 170
INTERP_DIR, 170
INTERP_ENG, 171
INTERP_STD, 171
ISPT, 171
MAT_BSR, 171
MAT_CSR, 171
MAT_CSRL, 171
MAT_FREE, 172
MAT_STR, 172
MAT_SymCSR, 172
MAT_bBSR, 171
MAT_bCSR, 171
MAX_AMG_LVL, 172
MAX_CRATE, 172
MAX_REFINE_LVL, 172
MAX_RESTART, 172
MAX_STAG, 172
MIN_CDOF, 173
MIN_CRATE, 173
NL_AMLI_CYCLE, 173
NO_ORDER, 173
OFF, 173
ON, 173
OPENMP_HOLDS, 173
PAIRWISE, 173
PREC_AMG, 174
PREC_DIAG, 174
PREC_FMG, 174
PREC_ILU, 174
PREC_NULL, 174
PREC_SCHWARZ, 174
PRINT_ALL, 174
PRINT_MIN, 174
PRINT_MORE, 175
PRINT_MOST, 175
PRINT_NONE, 175
PRINT_SOME, 175
SA_AMG, 175
SCHWARZ_BACKWARD, 175
SCHWARZ_FORWARD, 175
SCHWARZ_SYMMETRIC, 175
SMALLREAL, 176
SMALLREAL2, 176
SMOOTHER_BLKOil, 176
SMOOTHER_CG, 176
SMOOTHER_GS, 176
SMOOTHER_GSOR, 176
SMOOTHER_JACOBI, 176
SMOOTHER_L1DIAG, 176
SMOOTHER_POLY, 177
SMOOTHER_SGS, 177
SMOOTHER_SGSOR, 177
SMOOTHER_SOR, 177
SMOOTHER_SPETEN, 177
SMOOTHER_SSOR, 177
SOLVER_AMG, 177
SOLVER_BiCGstab, 177
SOLVER_CG, 177
SOLVER_DEFAULT, 178
SOLVER_FMG, 178
SOLVER_GCG, 178
SOLVER_GCR, 178
SOLVER_GMRES, 178
SOLVER_MUMPS, 178
SOLVER_MinRes, 178
SOLVER_SBiCGstab, 178
SOLVER_SCG, 179
SOLVER_SGCG, 179
SOLVER_SGMRES, 179

- SOLVER_SMinRes, [179](#)
- SOLVER_SUPERLU, [179](#)
- SOLVER_SVFGMRES, [179](#)
- SOLVER_SVGMRES, [179](#)
- SOLVER_UMFPACK, [179](#)
- SOLVER_VFGMRES, [179](#)
- SOLVER_VGMRES, [180](#)
- STAG_RATIO, [180](#)
- STOP_MOD_REL_RES, [180](#)
- STOP_REL_PRECRES, [180](#)
- STOP_REL_RES, [180](#)
- TRUE, [180](#)
- UA_AMG, [180](#)
- UNPT, [180](#)
- USERDEFINED, [181](#)
- V_CYCLE, [181](#)
- VMB, [181](#)
- W_CYCLE, [181](#)
- fasp_dbsr_Linfinity_dcsr
 - sparse_block.c, [392](#)
- fasp_dbsr_alloc
 - sparse_bsr.c, [394](#)
- fasp_dbsr_cp
 - sparse_bsr.c, [395](#)
- fasp_dbsr_create
 - sparse_bsr.c, [395](#)
- fasp_dbsr_diagLU
 - sparse_bsr.c, [398](#)
- fasp_dbsr_diagLU2
 - sparse_bsr.c, [398](#)
- fasp_dbsr_diaginv
 - sparse_bsr.c, [396](#)
- fasp_dbsr_diaginv2
 - sparse_bsr.c, [396](#)
- fasp_dbsr_diaginv3
 - sparse_bsr.c, [397](#)
- fasp_dbsr_diaginv4
 - sparse_bsr.c, [397](#)
- fasp_dbsr_diagpref
 - sparse_bsr.c, [399](#)
- fasp_dbsr_free
 - sparse_bsr.c, [399](#)
- fasp_dbsr_getblk
 - sparse_block.c, [391](#)
- fasp_dbsr_getblk_dcsr
 - sparse_block.c, [392](#)
- fasp_dbsr_getdiag
 - sparse_bsr.c, [400](#)
- fasp_dbsr_getdiaginv
 - sparse_bsr.c, [400](#)
- fasp_dbsr_null
 - sparse_bsr.c, [401](#)
- fasp_dbsr_plot
 - graphics.c, [197](#)
- fasp_dbsr_print
 - io.c, [222](#)
- fasp_dbsr_read
 - io.c, [223](#)
- fasp_dbsr_subplot
 - graphics.c, [197](#)
- fasp_dbsr_trans
 - sparse_bsr.c, [401](#)
- fasp_dbsr_write
 - io.c, [223](#)
- fasp_dbsr_write_coo
 - io.c, [224](#)
- fasp_dcoo1_read
 - io.c, [224](#)
- fasp_dcoo_alloc
 - sparse_coo.c, [402](#)
- fasp_dcoo_create
 - sparse_coo.c, [402](#)
- fasp_dcoo_free
 - sparse_coo.c, [403](#)
- fasp_dcoo_print
 - io.c, [225](#)
- fasp_dcoo_read
 - io.c, [225](#)
- fasp_dcoo_shift
 - sparse_coo.c, [403](#)
- fasp_dcoo_shift_read
 - io.c, [226](#)
- fasp_dcoo_write
 - io.c, [226](#)
- fasp_dcsr_CMK_order
 - ordering.c, [278](#)
- fasp_dcsr_RCMK_order
 - ordering.c, [278](#)
- fasp_dcsr_Schwarz_backward_smoother
 - schwarz_setup.c, [354](#)
- fasp_dcsr_Schwarz_forward_smoother
 - schwarz_setup.c, [354](#)
- fasp_dcsr_alloc
 - sparse_csr.c, [405](#)
- fasp_dcsr_compress
 - sparse_csr.c, [405](#)
- fasp_dcsr_compress_inplace
 - sparse_csr.c, [406](#)
- fasp_dcsr_cp
 - sparse_csr.c, [406](#)
- fasp_dcsr_create
 - sparse_csr.c, [407](#)
- fasp_dcsr_diagpref
 - sparse_csr.c, [407](#)
- fasp_dcsr_eig
 - eigen.c, [149](#)
- fasp_dcsr_free
 - sparse_csr.c, [408](#)

fasp_dcsr_getblk
 sparse_block.c, 393
 fasp_dcsr_getcol
 sparse_csr.c, 408
 fasp_dcsr_getdiag
 sparse_csr.c, 408
 fasp_dcsr_multicoloring
 sparse_csr.c, 409
 fasp_dcsr_null
 sparse_csr.c, 409
 fasp_dcsr_perm
 sparse_csr.c, 409
 fasp_dcsr_plot
 graphics.c, 197
 fasp_dcsr_print
 io.c, 227
 fasp_dcsr_read
 io.c, 227
 fasp_dcsr_regdiag
 sparse_csr.c, 411
 fasp_dcsr_shift
 sparse_csr.c, 411
 fasp_dcsr_sort
 sparse_csr.c, 412
 fasp_dcsr_subplot
 graphics.c, 198
 fasp_dcsr_symdiagscale
 sparse_csr.c, 412
 fasp_dcsr_sympat
 sparse_csr.c, 412
 fasp_dcsr_trans
 sparse_csr.c, 414
 fasp_dcsr_write_coo
 io.c, 227
 fasp_dcsrl_create
 sparse_csrl.c, 418
 fasp_dcsrl_free
 sparse_csrl.c, 419
 fasp_dcsrvec1_read
 io.c, 228
 fasp_dcsrvec1_write
 io.c, 228
 fasp_dcsrvec2_read
 io.c, 229
 fasp_dcsrvec2_write
 io.c, 230
 fasp_dmtx_read
 io.c, 230
 fasp_dmtxsym_read
 io.c, 231
 fasp_dstr_alloc
 sparse_str.c, 420
 fasp_dstr_cp
 sparse_str.c, 420
 fasp_dstr_create
 sparse_str.c, 421
 fasp_dstr_diagscale
 blas_str.c, 132
 fasp_dstr_free
 sparse_str.c, 421
 fasp_dstr_null
 sparse_str.c, 421
 fasp_dstr_print
 io.c, 231
 fasp_dstr_read
 io.c, 232
 fasp_dstr_write
 io.c, 232
 fasp_dvec_alloc
 vec.c, 452
 fasp_dvec_cp
 vec.c, 452
 fasp_dvec_create
 vec.c, 453
 fasp_dvec_free
 vec.c, 453
 fasp_dvec_isnan
 vec.c, 453
 fasp_dvec_maxdiff
 vec.c, 455
 fasp_dvec_null
 vec.c, 455
 fasp_dvec_print
 io.c, 233
 fasp_dvec_rand
 vec.c, 456
 fasp_dvec_read
 io.c, 233
 fasp_dvec_set
 vec.c, 456
 fasp_dvec_symdiagscale
 vec.c, 457
 fasp_dvec_write
 io.c, 234
 fasp_dvecind_read
 io.c, 234
 fasp_dvecind_write
 io.c, 234
 fasp_famg_solve
 amg_solve.c, 79
 fasp_format_bdcsr_dcsr
 formats.c, 183
 fasp_format_dbsr_dcoo
 formats.c, 183
 fasp_format_dbsr_dcsr
 formats.c, 184
 fasp_format_dcoo_dcsr
 formats.c, 184

fasp_format_dcsr_dbsr
formats.c, [184](#)

fasp_format_dcsr_dcoo
formats.c, [186](#)

fasp_format_dcsr_dcsr
formats.c, [186](#)

fasp_format_dstr_dbsr
formats.c, [187](#)

fasp_format_dstr_dcsr
formats.c, [187](#)

fasp_fwrapper_amg_
wrapper.c, [461](#)

fasp_fwrapper_krylov_amg_
wrapper.c, [462](#)

fasp_gauss2d
quadrature.c, [351](#)

fasp_generate_diaginv_block
smoother_str.c, [383](#)

fasp_gettime
timing.c, [451](#)

fasp_grid2d_plot
graphics.c, [198](#)

fasp_hb_read
io.c, [236](#)

fasp_iarray_cp
array.c, [85](#)

fasp_iarray_set
array.c, [85](#)

fasp_icsr_cp
sparse_csr.c, [414](#)

fasp_icsr_create
sparse_csr.c, [414](#)

fasp_icsr_free
sparse_csr.c, [416](#)

fasp_icsr_null
sparse_csr.c, [416](#)

fasp_icsr_trans
sparse_csr.c, [416](#)

fasp_iden_free
smat.c, [360](#)

fasp_ilu_data_alloc
init.c, [208](#)

fasp_ilu_data_free
init.c, [208](#)

fasp_ilu_data_null
init.c, [208](#)

fasp_ilu_dbsr_setup
ilu_setup_bsr.c, [201](#)

fasp_ilu_dcsr_setup
ilu_setup_csr.c, [202](#)

fasp_ilu_dstr_setup0
ilu_setup_str.c, [203](#)

fasp_ilu_dstr_setup1
ilu_setup_str.c, [203](#)

fasp_ivec_alloc
vec.c, [457](#)

fasp_ivec_create
vec.c, [457](#)

fasp_ivec_free
vec.c, [459](#)

fasp_ivec_print
io.c, [236](#)

fasp_ivec_read
io.c, [237](#)

fasp_ivec_set
vec.c, [459](#)

fasp_ivec_write
io.c, [237](#)

fasp_ivecind_read
io.c, [238](#)

fasp_matrix_read
io.c, [238](#)

fasp_matrix_read_bin
io.c, [239](#)

fasp_matrix_write
io.c, [239](#)

fasp_mem_calloc
memory.c, [265](#)

fasp_mem_check
memory.c, [265](#)

fasp_mem_dcsr_check
memory.c, [265](#)

fasp_mem_free
memory.c, [266](#)

fasp_mem_iludata_check
memory.c, [266](#)

fasp_mem_realloc
memory.c, [267](#)

fasp_mem_usage
memory.c, [267](#)

fasp_param_Schwarz_init
parameters.c, [284](#)

fasp_param_Schwarz_print
parameters.c, [284](#)

fasp_param_Schwarz_set
parameters.c, [286](#)

fasp_param_amg_init
parameters.c, [280](#)

fasp_param_amg_print
parameters.c, [280](#)

fasp_param_amg_set
parameters.c, [281](#)

fasp_param_amg_to_prec
parameters.c, [281](#)

fasp_param_amg_to_prec_bsr
parameters.c, [281](#)

fasp_param_check
input.c, [211](#)

fasp_param_ilu_init
 parameters.c, [282](#)
 fasp_param_ilu_print
 parameters.c, [282](#)
 fasp_param_ilu_set
 parameters.c, [282](#)
 fasp_param_init
 parameters.c, [283](#)
 fasp_param_input
 input.c, [212](#)
 fasp_param_input_init
 parameters.c, [283](#)
 fasp_param_prec_to_amg
 parameters.c, [283](#)
 fasp_param_prec_to_amg_bsr
 parameters.c, [284](#)
 fasp_param_set
 parameters.c, [286](#)
 fasp_param_solver_init
 parameters.c, [286](#)
 fasp_param_solver_print
 parameters.c, [287](#)
 fasp_param_solver_set
 parameters.c, [287](#)
 fasp_poisson_fgmg_1D
 gmh_poisson.c, [190](#)
 fasp_poisson_fgmg_2D
 gmh_poisson.c, [190](#)
 fasp_poisson_fgmg_3D
 gmh_poisson.c, [190](#)
 fasp_poisson_gmg_1D
 gmh_poisson.c, [192](#)
 fasp_poisson_gmg_2D
 gmh_poisson.c, [192](#)
 fasp_poisson_gmg_3D
 gmh_poisson.c, [193](#)
 fasp_poisson_pcg_gmg_1D
 gmh_poisson.c, [193](#)
 fasp_poisson_pcg_gmg_2D
 gmh_poisson.c, [195](#)
 fasp_poisson_pcg_gmg_3D
 gmh_poisson.c, [195](#)
 fasp_precond_Schwarz
 precond_csr.c, [335](#)
 fasp_precond_amg
 precond_csr.c, [331](#)
 fasp_precond_amg_nk
 precond_csr.c, [332](#)
 fasp_precond_amli
 precond_csr.c, [332](#)
 fasp_precond_block_SGS_3
 precond_bcsr.c, [319](#)
 fasp_precond_block_SGS_3_amg
 precond_bcsr.c, [321](#)
 fasp_precond_block_diag_3
 precond_bcsr.c, [316](#)
 fasp_precond_block_diag_3_amg
 precond_bcsr.c, [317](#)
 fasp_precond_block_diag_4
 precond_bcsr.c, [317](#)
 fasp_precond_block_lower_3
 precond_bcsr.c, [317](#)
 fasp_precond_block_lower_3_amg
 precond_bcsr.c, [319](#)
 fasp_precond_block_lower_4
 precond_bcsr.c, [319](#)
 fasp_precond_block_upper_3
 precond_bcsr.c, [321](#)
 fasp_precond_block_upper_3_amg
 precond_bcsr.c, [321](#)
 fasp_precond_data_null
 init.c, [210](#)
 fasp_precond_dbsr_amg
 precond_bsr.c, [324](#)
 fasp_precond_dbsr_amg_nk
 precond_bsr.c, [324](#)
 fasp_precond_dbsr_diag
 precond_bsr.c, [325](#)
 fasp_precond_dbsr_diag_nc2
 precond_bsr.c, [325](#)
 fasp_precond_dbsr_diag_nc3
 precond_bsr.c, [326](#)
 fasp_precond_dbsr_diag_nc5
 precond_bsr.c, [326](#)
 fasp_precond_dbsr_diag_nc7
 precond_bsr.c, [328](#)
 fasp_precond_dbsr_ilu
 precond_bsr.c, [328](#)
 fasp_precond_dbsr_nl_amli
 precond_bsr.c, [330](#)
 fasp_precond_diag
 precond_csr.c, [332](#)
 fasp_precond_dstr_blockgs
 precond_str.c, [338](#)
 fasp_precond_dstr_diag
 precond_str.c, [338](#)
 fasp_precond_dstr_ilu0
 precond_str.c, [338](#)
 fasp_precond_dstr_ilu0_backward
 precond_str.c, [339](#)
 fasp_precond_dstr_ilu0_forward
 precond_str.c, [339](#)
 fasp_precond_dstr_ilu1
 precond_str.c, [339](#)
 fasp_precond_dstr_ilu1_backward
 precond_str.c, [341](#)
 fasp_precond_dstr_ilu1_forward
 precond_str.c, [341](#)

fasp_precond_famg
 precond_csr.c, [333](#)
 fasp_precond_free
 precond_csr.c, [333](#)
 fasp_precond_ilu
 precond_csr.c, [333](#)
 fasp_precond_ilu_backward
 precond_csr.c, [334](#)
 fasp_precond_ilu_forward
 precond_csr.c, [334](#)
 fasp_precond_nl_amli
 precond_csr.c, [335](#)
 fasp_precond_null
 init.c, [210](#)
 fasp_precond_setup
 precond_csr.c, [335](#)
 fasp_precond_sweeping
 precond_bcsr.c, [323](#)
 fasp_quad2d
 quadrature.c, [352](#)
 fasp_set_GS_threads
 threads.c, [448](#)
 fasp_smat_identity
 smat.c, [360](#)
 fasp_smat_identity_nc2
 smat.c, [362](#)
 fasp_smat_identity_nc3
 smat.c, [362](#)
 fasp_smat_identity_nc5
 smat.c, [362](#)
 fasp_smat_identity_nc7
 smat.c, [363](#)
 fasp_smat_lu_decomp
 lu.c, [262](#)
 fasp_smat_lu_solve
 lu.c, [263](#)
 fasp_smoother_dbsr_gs
 smoother_bsr.c, [364](#)
 fasp_smoother_dbsr_gs1
 smoother_bsr.c, [365](#)
 fasp_smoother_dbsr_gs_ascend
 smoother_bsr.c, [365](#)
 fasp_smoother_dbsr_gs_ascend1
 smoother_bsr.c, [366](#)
 fasp_smoother_dbsr_gs_descend
 smoother_bsr.c, [366](#)
 fasp_smoother_dbsr_gs_descend1
 smoother_bsr.c, [367](#)
 fasp_smoother_dbsr_gs_order1
 smoother_bsr.c, [367](#)
 fasp_smoother_dbsr_gs_order2
 smoother_bsr.c, [368](#)
 fasp_smoother_dbsr_ilu
 smoother_bsr.c, [368](#)
 fasp_smoother_dbsr_jacobi
 smoother_bsr.c, [369](#)
 fasp_smoother_dbsr_jacobi1
 smoother_bsr.c, [369](#)
 fasp_smoother_dbsr_jacobi_setup
 smoother_bsr.c, [370](#)
 fasp_smoother_dbsr_sor
 smoother_bsr.c, [370](#)
 fasp_smoother_dbsr_sor1
 smoother_bsr.c, [371](#)
 fasp_smoother_dbsr_sor_ascend
 smoother_bsr.c, [371](#)
 fasp_smoother_dbsr_sor_descend
 smoother_bsr.c, [372](#)
 fasp_smoother_dbsr_sor_order
 smoother_bsr.c, [372](#)
 fasp_smoother_dcsr_L1diag
 smoother_csr.c, [377](#)
 fasp_smoother_dcsr_gs
 smoother_csr.c, [374](#)
 fasp_smoother_dcsr_gs_cf
 smoother_csr.c, [374](#)
 fasp_smoother_dcsr_gs_rb3d
 smoother_csr.c, [375](#)
 fasp_smoother_dcsr_gscr
 smoother_csr_cr.c, [379](#)
 fasp_smoother_dcsr_ilu
 smoother_csr.c, [375](#)
 fasp_smoother_dcsr_jacobi
 smoother_csr.c, [376](#)
 fasp_smoother_dcsr_kaczmarz
 smoother_csr.c, [376](#)
 fasp_smoother_dcsr_poly
 smoother_csr_poly.c, [381](#)
 fasp_smoother_dcsr_poly_old
 smoother_csr_poly.c, [381](#)
 fasp_smoother_dcsr_sgs
 smoother_csr.c, [377](#)
 fasp_smoother_dcsr_sor
 smoother_csr.c, [378](#)
 fasp_smoother_dcsr_sor_cf
 smoother_csr.c, [378](#)
 fasp_smoother_dstr_gs
 smoother_str.c, [383](#)
 fasp_smoother_dstr_gs1
 smoother_str.c, [383](#)
 fasp_smoother_dstr_gs_ascend
 smoother_str.c, [384](#)
 fasp_smoother_dstr_gs_cf
 smoother_str.c, [384](#)
 fasp_smoother_dstr_gs_descend
 smoother_str.c, [385](#)
 fasp_smoother_dstr_gs_order
 smoother_str.c, [385](#)

fasp_smoother_dstr_jacobi
 smoother_str.c, [386](#)
 fasp_smoother_dstr_jacobi1
 smoother_str.c, [386](#)
 fasp_smoother_dstr_schwarz
 smoother_str.c, [387](#)
 fasp_smoother_dstr_sor
 smoother_str.c, [387](#)
 fasp_smoother_dstr_sor1
 smoother_str.c, [388](#)
 fasp_smoother_dstr_sor_ascend
 smoother_str.c, [388](#)
 fasp_smoother_dstr_sor_cf
 smoother_str.c, [389](#)
 fasp_smoother_dstr_sor_descend
 smoother_str.c, [389](#)
 fasp_smoother_dstr_sor_order
 smoother_str.c, [390](#)
 fasp_solver_amg
 amg.c, [69](#)
 fasp_solver_amli
 amlirecur.c, [80](#)
 fasp_solver_bdcscr_itsolver
 itsolver_bcsr.c, [244](#)
 fasp_solver_bdcscr_krylov
 itsolver_bcsr.c, [245](#)
 fasp_solver_bdcscr_krylov_block_3
 itsolver_bcsr.c, [245](#)
 fasp_solver_bdcscr_krylov_block_4
 itsolver_bcsr.c, [246](#)
 fasp_solver_bdcscr_krylov_sweeping
 itsolver_bcsr.c, [246](#)
 fasp_solver_bdcscr_pbcgs
 pbcgs.c, [289](#)
 fasp_solver_bdcscr_pcg
 pcg.c, [295](#)
 fasp_solver_bdcscr_pgmres
 pgmres.c, [305](#)
 fasp_solver_bdcscr_pminres
 pminres.c, [311](#)
 fasp_solver_bdcscr_pvfgmres
 pvfgmres.c, [342](#)
 fasp_solver_bdcscr_pvgmres
 pvgmres.c, [346](#)
 fasp_solver_bdcscr_spbcs
 spbcgs.c, [430](#)
 fasp_solver_bdcscr_spcg
 spcg.c, [435](#)
 fasp_solver_bdcscr_spgmres
 spgmres.c, [438](#)
 fasp_solver_bdcscr_spminres
 spminres.c, [442](#)
 fasp_solver_bdcscr_spvgmres
 spvgmres.c, [444](#)
 fasp_solver_dbsr_itsolver
 itsolver_bsr.c, [248](#)
 fasp_solver_dbsr_krylov
 itsolver_bsr.c, [248](#)
 fasp_solver_dbsr_krylov_amg
 itsolver_bsr.c, [249](#)
 fasp_solver_dbsr_krylov_amg_nk
 itsolver_bsr.c, [249](#)
 fasp_solver_dbsr_krylov_diag
 itsolver_bsr.c, [250](#)
 fasp_solver_dbsr_krylov_ilu
 itsolver_bsr.c, [250](#)
 fasp_solver_dbsr_krylov_nk_amg
 itsolver_bsr.c, [251](#)
 fasp_solver_dbsr_pbcgs
 pbcgs.c, [290](#)
 fasp_solver_dbsr_pcg
 pcg.c, [296](#)
 fasp_solver_dbsr_pgmres
 pgmres.c, [306](#)
 fasp_solver_dbsr_pvfgmres
 pvfgmres.c, [343](#)
 fasp_solver_dbsr_pvgmres
 pvgmres.c, [347](#)
 fasp_solver_dbsr_spbcs
 spbcgs.c, [431](#)
 fasp_solver_dbsr_spgmres
 spgmres.c, [438](#)
 fasp_solver_dbsr_spvgmres
 spvgmres.c, [445](#)
 fasp_solver_dcsr_itsolver
 itsolver_csr.c, [252](#)
 fasp_solver_dcsr_krylov
 itsolver_csr.c, [253](#)
 fasp_solver_dcsr_krylov_Schwarz
 itsolver_csr.c, [256](#)
 fasp_solver_dcsr_krylov_amg
 itsolver_csr.c, [253](#)
 fasp_solver_dcsr_krylov_amg_nk
 itsolver_csr.c, [254](#)
 fasp_solver_dcsr_krylov_diag
 itsolver_csr.c, [254](#)
 fasp_solver_dcsr_krylov_ilu
 itsolver_csr.c, [255](#)
 fasp_solver_dcsr_krylov_ilu_M
 itsolver_csr.c, [255](#)
 fasp_solver_dcsr_pbcgs
 pbcgs.c, [290](#)
 fasp_solver_dcsr_pcg
 pcg.c, [296](#)
 fasp_solver_dcsr_pgcr
 pgcr.c, [300](#)
 fasp_solver_dcsr_pgcr
 pgcr.c, [303](#)

fasp_solver_dcsr_pgcr1
pgcr.c, 304

fasp_solver_dcsr_pgmres
pgmres.c, 307

fasp_solver_dcsr_pminres
pminres.c, 311

fasp_solver_dcsr_pvfgrmres
pvfgrmres.c, 344

fasp_solver_dcsr_pvgmres
pvgmres.c, 347

fasp_solver_dcsr_spgbcs
spgbcs.c, 431

fasp_solver_dcsr_spcg
spcg.c, 436

fasp_solver_dcsr_spgmres
spgmres.c, 439

fasp_solver_dcsr_spmminres
spmminres.c, 442

fasp_solver_dcsr_spvgrmres
spvgrmres.c, 445

fasp_solver_dstr_itsolver
itsolver_str.c, 259

fasp_solver_dstr_krylov
itsolver_str.c, 260

fasp_solver_dstr_krylov_blockgs
itsolver_str.c, 260

fasp_solver_dstr_krylov_diag
itsolver_str.c, 261

fasp_solver_dstr_krylov_ilu
itsolver_str.c, 261

fasp_solver_dstr_pbcgs
pbcgs.c, 291

fasp_solver_dstr_pcg
pcg.c, 297

fasp_solver_dstr_pgmres
pgmres.c, 307

fasp_solver_dstr_pminres
pminres.c, 312

fasp_solver_dstr_pvgmres
pvgmres.c, 349

fasp_solver_dstr_spgbcs
spgbcs.c, 433

fasp_solver_dstr_spcg
spcg.c, 436

fasp_solver_dstr_spgmres
spgmres.c, 439

fasp_solver_dstr_spmminres
spmminres.c, 443

fasp_solver_dstr_spvgrmres
spvgrmres.c, 447

fasp_solver_famg
famg.c, 151

fasp_solver_fmngcycle
fmngcycle.c, 182

fasp_solver_itsolver
itsolver_mf.c, 257

fasp_solver_itsolver_init
itsolver_mf.c, 258

fasp_solver_krylov
itsolver_mf.c, 258

fasp_solver_mngcycle
mngcycle.c, 271

fasp_solver_mngcycle_bsr
mngcycle.c, 272

fasp_solver_mngrecur
mngrecur.c, 273

fasp_solver_mumps
interface_mumps.c, 213

fasp_solver_mumps_steps
interface_mumps.c, 214

fasp_solver_nl_amli
amlirecur.c, 81

fasp_solver_nl_amli_bsr
amlirecur.c, 81

fasp_solver_pbcgs
pbcgs_mf.c, 293

fasp_solver_pcg
pcg_mf.c, 299

fasp_solver_pgcg
pgcg_mf.c, 301

fasp_solver_pgmres
pgmres_mf.c, 309

fasp_solver_pminres
pminres_mf.c, 314

fasp_solver_pvfgrmres
pvfgrmres_mf.c, 345

fasp_solver_pvgmres
pvgmres_mf.c, 350

fasp_solver_superlu
interface_superlu.c, 216

fasp_solver_umfpack
interface_umfpack.c, 217

fasp_sparse_MIS
sparse_util.c, 425

fasp_sparse_aat
sparse_util.c, 423

fasp_sparse_abyb
sparse_util.c, 423

fasp_sparse_abybms
sparse_util.c, 424

fasp_sparse_aplbms
sparse_util.c, 424

fasp_sparse_aplusb
sparse_util.c, 425

fasp_sparse_iit
sparse_util.c, 425

fasp_sparse_rapcmp
sparse_util.c, 426

- fasp_sparse_rapms_
 - sparse_util.c, [426](#)
 - fasp_sparse_wta_
 - sparse_util.c, [427](#)
 - fasp_sparse_wtams_
 - sparse_util.c, [428](#)
 - fasp_sparse_ytx_
 - sparse_util.c, [428](#)
 - fasp_sparse_ytxbig_
 - sparse_util.c, [428](#)
 - fasp_vector_read
 - io.c, [241](#)
 - fasp_vector_write
 - io.c, [242](#)
 - fasp_wrapper_dbsr_krylov_amg
 - wrapper.c, [462](#)
 - fasp_wrapper_dcoo_dbsr_krylov_amg
 - wrapper.c, [463](#)
 - fmgcycle.c, [181](#)
 - fasp_solver_fmgcycle, [182](#)
 - formats.c, [182](#)
 - fasp_format_bdcsl_dcsr, [183](#)
 - fasp_format_dbsr_dcoo, [183](#)
 - fasp_format_dbsr_dcsr, [184](#)
 - fasp_format_dcoo_dcsr, [184](#)
 - fasp_format_dcsr_dbsr, [184](#)
 - fasp_format_dcsr_dcoo, [186](#)
 - fasp_format_dcsrl_dcsr, [186](#)
 - fasp_format_dstr_dbsr, [187](#)
 - fasp_format_dstr_dcsr, [187](#)
 - G0PT
 - fasp_const.h, [170](#)
 - GE
 - fasp.h, [155](#)
 - GT
 - fasp.h, [155](#)
 - givens.c, [188](#)
 - fasp_aux_givens, [188](#)
 - gmh_poisson.c, [189](#)
 - fasp_poisson_fgmg_1D, [190](#)
 - fasp_poisson_fgmg_2D, [190](#)
 - fasp_poisson_fgmg_3D, [190](#)
 - fasp_poisson_gmg_1D, [192](#)
 - fasp_poisson_gmg_2D, [192](#)
 - fasp_poisson_gmg_3D, [193](#)
 - fasp_poisson_pcg_gmg_1D, [193](#)
 - fasp_poisson_pcg_gmg_2D, [195](#)
 - fasp_poisson_pcg_gmg_3D, [195](#)
 - graphics.c, [196](#)
 - fasp_dbsr_plot, [197](#)
 - fasp_dbsr_subplot, [197](#)
 - fasp_dcsr_plot, [197](#)
 - fasp_dcsr_subplot, [198](#)
 - fasp_grid2d_plot, [198](#)
 - grid2d, [32](#)
 - e, [33](#)
 - edges, [33](#)
 - ediri, [33](#)
 - efather, [33](#)
 - fasp.h, [157](#)
 - p, [33](#)
 - pdiri, [33](#)
 - pfather, [34](#)
 - s, [34](#)
 - t, [34](#)
 - tfather, [34](#)
 - triangles, [34](#)
 - vertices, [34](#)
 - iCOOmat, [34](#)
 - fasp.h, [157](#)
 - iCSRmat, [35](#)
 - fasp.h, [157](#)
 - ILU_data, [36](#)
 - ILU_droptol
 - input_param, [43](#)
 - ILU_lfil
 - input_param, [43](#)
 - ILU_param, [37](#)
 - ILU_permtol
 - input_param, [43](#)
 - ILU_relax
 - input_param, [43](#)
 - ILU_type
 - input_param, [43](#)
 - ILUk
 - fasp_const.h, [170](#)
 - ILUt
 - fasp_const.h, [170](#)
 - ILUtp
 - fasp_const.h, [170](#)
 - IMAP
 - fasp.h, [158](#)
 - INT
 - fasp.h, [155](#)
 - INTERP_DIR
 - fasp_const.h, [170](#)
 - INTERP_ENG
 - fasp_const.h, [171](#)
 - INTERP_STD
 - fasp_const.h, [171](#)
 - ISNAN
 - fasp.h, [155](#)
 - ISPT
 - fasp_const.h, [171](#)
 - idenmat, [36](#)
 - fasp.h, [158](#)

- ilength
 - io.c, [243](#)
- ilu.f, [200](#)
- ilu_setup_bsr.c, [200](#)
 - fasp_ilu_dbsr_setup, [201](#)
- ilu_setup_csr.c, [201](#)
 - fasp_ilu_dcsr_setup, [202](#)
- ilu_setup_str.c, [202](#)
 - fasp_ilu_dstr_setup0, [203](#)
 - fasp_ilu_dstr_setup1, [203](#)
- inifile
 - input_param, [43](#)
- init.c, [204](#)
 - fasp_Schwarz_data_free, [210](#)
 - fasp_amg_data_bsr_create, [205](#)
 - fasp_amg_data_bsr_free, [206](#)
 - fasp_amg_data_create, [206](#)
 - fasp_amg_data_free, [206](#)
 - fasp_ilu_data_alloc, [208](#)
 - fasp_ilu_data_free, [208](#)
 - fasp_ilu_data_null, [208](#)
 - fasp_precond_data_null, [210](#)
 - fasp_precond_null, [210](#)
- input.c, [211](#)
 - fasp_param_check, [211](#)
 - fasp_param_input, [212](#)
- input_param, [38](#)
 - AMG_ILU_levels, [40](#)
 - AMG_Schwarz_levels, [41](#)
 - AMG_aggregation_type, [39](#)
 - AMG_aggressive_level, [39](#)
 - AMG_aggressive_path, [39](#)
 - AMG_amli_degree, [39](#)
 - AMG_coarse_dof, [39](#)
 - AMG_coarse_scaling, [40](#)
 - AMG_coarse_solver, [40](#)
 - AMG_coarsening_type, [40](#)
 - AMG_cycle_type, [40](#)
 - AMG_interpolation_type, [40](#)
 - AMG_levels, [40](#)
 - AMG_max_aggregation, [40](#)
 - AMG_max_row_sum, [40](#)
 - AMG_maxit, [41](#)
 - AMG_nl_amli_krylov_type, [41](#)
 - AMG_pair_number, [41](#)
 - AMG_polynomial_degree, [41](#)
 - AMG_postsmooth_iter, [41](#)
 - AMG_presmooth_iter, [41](#)
 - AMG_quality_bound, [41](#)
 - AMG_relaxation, [41](#)
 - AMG_smooth_filter, [42](#)
 - AMG_smooth_order, [42](#)
 - AMG_smoother, [42](#)
 - AMG_strong_coupled, [42](#)
 - AMG_strong_threshold, [42](#)
 - AMG_tentative_smooth, [42](#)
 - AMG_tol, [42](#)
 - AMG_truncation_threshold, [42](#)
 - AMG_type, [42](#)
 - ILU_droptol, [43](#)
 - ILU_lfil, [43](#)
 - ILU_permtol, [43](#)
 - ILU_relax, [43](#)
 - ILU_type, [43](#)
 - inifile, [43](#)
 - itsolver_maxit, [43](#)
 - itsolver_tol, [43](#)
 - output_type, [43](#)
 - precond_type, [44](#)
 - print_level, [44](#)
 - problem_num, [44](#)
 - restart, [44](#)
 - Schwarz_blk solver, [44](#)
 - Schwarz_maxlvl, [44](#)
 - Schwarz_mmsize, [44](#)
 - Schwarz_type, [44](#)
 - solver_type, [44](#)
 - stop_type, [45](#)
 - workdir, [45](#)
- interface_mumps.c, [212](#)
 - fasp_solver_mumps, [213](#)
 - fasp_solver_mumps_steps, [214](#)
- interface_samg.c, [214](#)
 - dCSRmat2SAMGInput, [215](#)
 - dvector2SAMGInput, [215](#)
- interface_superlu.c, [216](#)
 - fasp_solver_superlu, [216](#)
- interface_umfpack.c, [216](#)
 - fasp_solver_umfpack, [217](#)
- interpolation.c, [217](#)
 - fasp_amg_interp, [218](#)
 - fasp_amg_interp1, [218](#)
 - fasp_amg_interp_trunc, [219](#)
- interpolation_em.c, [219](#)
 - fasp_amg_interp_em, [220](#)
- io.c, [220](#)
 - dlength, [243](#)
 - fasp_dbsr_print, [222](#)
 - fasp_dbsr_read, [223](#)
 - fasp_dbsr_write, [223](#)
 - fasp_dbsr_write_coo, [224](#)
 - fasp_dcoo1_read, [224](#)
 - fasp_dcoo_print, [225](#)
 - fasp_dcoo_read, [225](#)
 - fasp_dcoo_shift_read, [226](#)
 - fasp_dcoo_write, [226](#)
 - fasp_dcsr_print, [227](#)
 - fasp_dcsr_read, [227](#)

- fasp_dcsr_write_coo, 227
 - fasp_dcsrvec1_read, 228
 - fasp_dcsrvec1_write, 228
 - fasp_dcsrvec2_read, 229
 - fasp_dcsrvec2_write, 230
 - fasp_dmtx_read, 230
 - fasp_dmtxsym_read, 231
 - fasp_dstr_print, 231
 - fasp_dstr_read, 232
 - fasp_dstr_write, 232
 - fasp_dvec_print, 233
 - fasp_dvec_read, 233
 - fasp_dvec_write, 234
 - fasp_dvecind_read, 234
 - fasp_dvecind_write, 234
 - fasp_hb_read, 236
 - fasp_ivec_print, 236
 - fasp_ivec_read, 237
 - fasp_ivec_write, 237
 - fasp_ivecind_read, 238
 - fasp_matrix_read, 238
 - fasp_matrix_read_bin, 239
 - fasp_matrix_write, 239
 - fasp_vector_read, 241
 - fasp_vector_write, 242
 - ilength, 243
- itsolver_bcsr.c, 243
 - fasp_solver_bdcsr_itsolver, 244
 - fasp_solver_bdcsr_krylov, 245
 - fasp_solver_bdcsr_krylov_block_3, 245
 - fasp_solver_bdcsr_krylov_block_4, 246
 - fasp_solver_bdcsr_krylov_sweeping, 246
- itsolver_bsr.c, 247
 - fasp_solver_dbsr_itsolver, 248
 - fasp_solver_dbsr_krylov, 248
 - fasp_solver_dbsr_krylov_amg, 249
 - fasp_solver_dbsr_krylov_amg_nk, 249
 - fasp_solver_dbsr_krylov_diag, 250
 - fasp_solver_dbsr_krylov_ilu, 250
 - fasp_solver_dbsr_krylov_nk_amg, 251
- itsolver_csr.c, 252
 - fasp_solver_dcsr_itsolver, 252
 - fasp_solver_dcsr_krylov, 253
 - fasp_solver_dcsr_krylov_Schwarz, 256
 - fasp_solver_dcsr_krylov_amg, 253
 - fasp_solver_dcsr_krylov_amg_nk, 254
 - fasp_solver_dcsr_krylov_diag, 254
 - fasp_solver_dcsr_krylov_ilu, 255
 - fasp_solver_dcsr_krylov_ilu_M, 255
- itsolver_maxit
 - input_param, 43
- itsolver_mf.c, 257
 - fasp_solver_itsolver, 257
 - fasp_solver_itsolver_init, 258
 - fasp_solver_krylov, 258
 - itsolver_param, 45
 - itsolver_type, 45
 - maxit, 45
 - precond_type, 46
 - print_level, 46
 - restart, 46
 - stop_type, 46
 - tol, 46
- itsolver_str.c, 259
 - fasp_solver_dstr_itsolver, 259
 - fasp_solver_dstr_krylov, 260
 - fasp_solver_dstr_krylov_blockgs, 260
 - fasp_solver_dstr_krylov_diag, 261
 - fasp_solver_dstr_krylov_ilu, 261
- itsolver_tol
 - input_param, 43
- itsolver_type
 - itsolver_param, 45
- ivector, 46
 - fasp.h, 158
- JA
 - dBSRmat, 28
- LE
 - fasp.h, 155
- LONG
 - fasp.h, 155
- LONGLONG
 - fasp.h, 155
- LS
 - fasp.h, 156
- LU_diag
 - precond_block_data, 50
- Link, 47
- LinkedList
 - fasp.h, 158
- linked_list, 47
- ListElement
 - fasp.h, 158
- local_A
 - precond_sweeping_data, 65
- local_LU
 - precond_sweeping_data, 65
- local_index
 - precond_sweeping_data, 65
- lu.c, 262
 - fasp_smat_lu_decomp, 262
 - fasp_smat_lu_solve, 263
- MAT_BSR
 - fasp_const.h, 171
- MAT_CSR
 - fasp_const.h, 171

- MAT_CSRL
 - fasp_const.h, [171](#)
- MAT_FREE
 - fasp_const.h, [172](#)
- MAT_STR
 - fasp_const.h, [172](#)
- MAT_SymCSR
 - fasp_const.h, [172](#)
- MAT_bBSR
 - fasp_const.h, [171](#)
- MAT_bCSR
 - fasp_const.h, [171](#)
- MAX
 - fasp.h, [156](#)
- MAX_AMG_LVL
 - fasp_const.h, [172](#)
- MAX_CRATE
 - fasp_const.h, [172](#)
- MAX_REFINE_LVL
 - fasp_const.h, [172](#)
- MAX_RESTART
 - fasp_const.h, [172](#)
- MAX_STAG
 - fasp_const.h, [172](#)
- MAXIMAP
 - fasp.h, [158](#)
- MIN
 - fasp.h, [156](#)
- MIN_CDOF
 - fasp_const.h, [173](#)
- MIN_CRATE
 - fasp_const.h, [173](#)
- maxit
 - itsolver_param, [45](#)
 - precond_FASP_blkoi_data, [62](#)
- memory.c, [264](#)
 - fasp_mem_calloc, [265](#)
 - fasp_mem_check, [265](#)
 - fasp_mem_dcsr_check, [265](#)
 - fasp_mem_free, [266](#)
 - fasp_mem_iludata_check, [266](#)
 - fasp_mem_realloc, [267](#)
 - fasp_mem_usage, [267](#)
 - total_alloc_count, [267](#)
 - total_alloc_mem, [267](#)
- message.c, [268](#)
 - fasp_chkerr, [268](#)
 - print_amgcomplexity, [269](#)
 - print_amgcomplexity_bsr, [269](#)
 - print_cputime, [269](#)
 - print_itinfo, [270](#)
 - print_message, [270](#)
- mgcycle.c, [271](#)
 - fasp_solver_mgcycle, [271](#)
 - fasp_solver_mgcycle_bsr, [272](#)
- mgl
 - precond_block_data, [50](#)
- mgl_data
 - precond_FASP_blkoi_data, [62](#)
- mgrecur.c, [272](#)
 - fasp_solver_mgrecur, [273](#)
- Mumps_data, [48](#)
- mxv_matfree, [48](#)
- NEDMALLOC
 - fasp.h, [156](#)
- NL_AMLI_CYCLE
 - fasp_const.h, [173](#)
- NO_ORDER
 - fasp_const.h, [173](#)
- neigh
 - precond_FASP_blkoi_data, [62](#)
- NumLayers
 - precond_sweeping_data, [65](#)
- nx_rb
 - fasp.h, [158](#)
- ny_rb
 - fasp.h, [159](#)
- nz_rb
 - fasp.h, [159](#)
- OFF
 - fasp_const.h, [173](#)
- ON
 - fasp_const.h, [173](#)
- OPENMP_HOLDS
 - fasp_const.h, [173](#)
- order
 - precond_FASP_blkoi_data, [62](#)
 - precond_block_reservoir_data, [52](#)
- ordering.c, [273](#)
 - fasp_BinarySearch, [277](#)
 - fasp_aux_dQuickSort, [274](#)
 - fasp_aux_dQuickSortIndex, [274](#)
 - fasp_aux_iQuickSort, [275](#)
 - fasp_aux_iQuickSortIndex, [275](#)
 - fasp_aux_merge, [276](#)
 - fasp_aux_msort, [276](#)
 - fasp_aux_unique, [277](#)
 - fasp_dcsr_CMK_order, [278](#)
 - fasp_dcsr_RCMK_order, [278](#)
- output_type
 - input_param, [43](#)
- p
 - grid2d, [33](#)
- PAIRWISE
 - fasp_const.h, [173](#)
- PP

- precond_FASP_blkoi_data, 63
- precond_block_reservoir_data, 53
- PREC_AMG
 - fasp_const.h, 174
- PREC_DIAG
 - fasp_const.h, 174
- PREC_FMG
 - fasp_const.h, 174
- PREC_ILU
 - fasp_const.h, 174
- PREC_NULL
 - fasp_const.h, 174
- PREC_SCHWARZ
 - fasp_const.h, 174
- PRINT_ALL
 - fasp_const.h, 174
- PRINT_MIN
 - fasp_const.h, 174
- PRINT_MORE
 - fasp_const.h, 175
- PRINT_MOST
 - fasp_const.h, 175
- PRINT_NONE
 - fasp_const.h, 175
- PRINT_SOME
 - fasp_const.h, 175
- PUT_INT
 - fasp.h, 156
- PUT_REAL
 - fasp.h, 156
- parameters.c, 279
 - fasp_param_Schwarz_init, 284
 - fasp_param_Schwarz_print, 284
 - fasp_param_Schwarz_set, 286
 - fasp_param_amg_init, 280
 - fasp_param_amg_print, 280
 - fasp_param_amg_set, 281
 - fasp_param_amg_to_prec, 281
 - fasp_param_amg_to_prec_bsr, 281
 - fasp_param_ilu_init, 282
 - fasp_param_ilu_print, 282
 - fasp_param_ilu_set, 282
 - fasp_param_init, 283
 - fasp_param_input_init, 283
 - fasp_param_prec_to_amg, 283
 - fasp_param_prec_to_amg_bsr, 284
 - fasp_param_set, 286
 - fasp_param_solver_init, 286
 - fasp_param_solver_print, 287
 - fasp_param_solver_set, 287
- pbcgs.c, 288
 - fasp_solver_bdcsr_pbcgs, 289
 - fasp_solver_dbsr_pbcgs, 290
 - fasp_solver_dcsr_pbcgs, 290
 - fasp_solver_dstr_pbcgs, 291
- pbcgs_mf.c, 292
 - fasp_solver_pbcgs, 293
- pcg.c, 294
 - fasp_solver_bdcsr_pcg, 295
 - fasp_solver_dbsr_pcg, 296
 - fasp_solver_dcsr_pcg, 296
 - fasp_solver_dstr_pcg, 297
- pcg_mf.c, 298
 - fasp_solver_pcg, 299
- pcgrid2d
 - fasp.h, 158
- pdiri
 - grid2d, 33
- perf_idx
 - precond_FASP_blkoi_data, 62
 - precond_block_reservoir_data, 53
- perf_neigh
 - precond_FASP_blkoi_data, 62
- pfather
 - grid2d, 34
- pgcg.c, 300
 - fasp_solver_dcsr_pgcg, 300
- pgcg_mf.c, 301
 - fasp_solver_pgcg, 301
- pgcr.c, 302
 - fasp_solver_dcsr_pgcr, 303
 - fasp_solver_dcsr_pgcr1, 304
- pgmres.c, 305
 - fasp_solver_bdcsr_pgmres, 305
 - fasp_solver_dbsr_pgmres, 306
 - fasp_solver_dcsr_pgmres, 307
 - fasp_solver_dstr_pgmres, 307
- pgmres_mf.c, 308
 - fasp_solver_pgmres, 309
- pgrid2d
 - fasp.h, 158
- pivot
 - precond_FASP_blkoi_data, 62
 - precond_block_reservoir_data, 53
- pivot_S
 - precond_FASP_blkoi_data, 62
- pivotS
 - precond_block_reservoir_data, 53
- pminres.c, 309
 - fasp_solver_bdcsr_pminres, 311
 - fasp_solver_dcsr_pminres, 311
 - fasp_solver_dstr_pminres, 312
- pminres_mf.c, 313
 - fasp_solver_pminres, 314
- precond, 49
 - precond_FASP_blkoi_data, 60
 - A, 61
 - diagin, 61

- diaginv_S, 62
- diaginv_noscale, 61
- maxit, 62
- mgl_data, 62
- neigh, 62
- order, 62
- PP, 63
- perf_idx, 62
- perf_neigh, 62
- pivot, 62
- pivot_S, 62
- r, 63
- RR, 63
- restart, 63
- SS, 63
- scaled, 63
- tol, 63
- w, 63
- WW, 64
- precond_bcsr.c, 315
 - fasp_precond_block_SGS_3, 319
 - fasp_precond_block_SGS_3_amg, 321
 - fasp_precond_block_diag_3, 316
 - fasp_precond_block_diag_3_amg, 317
 - fasp_precond_block_diag_4, 317
 - fasp_precond_block_lower_3, 317
 - fasp_precond_block_lower_3_amg, 319
 - fasp_precond_block_lower_4, 319
 - fasp_precond_block_upper_3, 321
 - fasp_precond_block_upper_3_amg, 321
 - fasp_precond_sweeping, 323
- precond_block_data, 49
 - A_diag, 50
 - Abcsr, 50
 - amgparam, 50
 - LU_diag, 50
 - mgl, 50
 - r, 50
- precond_block_reservoir_data, 51
 - diag, 52
 - diaginv, 52
 - diaginvS, 52
 - fasp_block.h, 161
 - order, 52
 - PP, 53
 - perf_idx, 53
 - pivot, 53
 - pivotS, 53
 - r, 53
 - RR, 53
 - SS, 53
 - scaled, 53
 - w, 53
 - WW, 54
- precond_bsr.c, 323
 - fasp_precond_dbsr_amg, 324
 - fasp_precond_dbsr_amg_nk, 324
 - fasp_precond_dbsr_diag, 325
 - fasp_precond_dbsr_diag_nc2, 325
 - fasp_precond_dbsr_diag_nc3, 326
 - fasp_precond_dbsr_diag_nc5, 326
 - fasp_precond_dbsr_diag_nc7, 328
 - fasp_precond_dbsr_ilu, 328
 - fasp_precond_dbsr_nl_amli, 330
- precond_csr.c, 330
 - fasp_precond_Schwarz, 335
 - fasp_precond_amg, 331
 - fasp_precond_amg_nk, 332
 - fasp_precond_amli, 332
 - fasp_precond_diag, 332
 - fasp_precond_famg, 333
 - fasp_precond_free, 333
 - fasp_precond_ilu, 333
 - fasp_precond_ilu_backward, 334
 - fasp_precond_ilu_forward, 334
 - fasp_precond_nl_amli, 335
 - fasp_precond_setup, 335
- precond_data, 54
- precond_data_bsr, 55
- precond_data_str, 57
- precond_diagbsr, 59
- precond_diagstr, 59
- precond_str.c, 337
 - fasp_precond_dstr_blockgs, 338
 - fasp_precond_dstr_diag, 338
 - fasp_precond_dstr_ilu0, 338
 - fasp_precond_dstr_ilu0_backward, 339
 - fasp_precond_dstr_ilu0_forward, 339
 - fasp_precond_dstr_ilu1, 339
 - fasp_precond_dstr_ilu1_backward, 341
 - fasp_precond_dstr_ilu1_forward, 341
- precond_sweeping_data, 64
 - A, 65
 - Ai, 65
 - local_A, 65
 - local_LU, 65
 - local_index, 65
 - NumLayers, 65
 - r, 65
 - w, 65
- precond_type
 - input_param, 44
 - itsolver_param, 46
- print_amgcomplexity
 - message.c, 269
- print_amgcomplexity_bsr
 - message.c, 269
- print_cputime

- message.c, 269
- print_itinfo
 - message.c, 270
- print_level
 - input_param, 44
 - itsolver_param, 46
- print_message
 - message.c, 270
- problem_num
 - input_param, 44
- pvfgmres.c, 342
 - fasp_solver_bdcsr_pvfgmres, 342
 - fasp_solver_dbsr_pvfgmres, 343
 - fasp_solver_dcsr_pvfgmres, 344
- pvfgmres_mf.c, 344
 - fasp_solver_pvfgmres, 345
- pvgmres.c, 346
 - fasp_solver_bdcsr_pvgmres, 346
 - fasp_solver_dbsr_pvgmres, 347
 - fasp_solver_dcsr_pvgmres, 347
 - fasp_solver_dstr_pvgmres, 349
- pvgmres_mf.c, 350
 - fasp_solver_pvgmres, 350
- quadrature.c, 351
 - fasp_gauss2d, 351
 - fasp_quad2d, 352
- r
 - precond_FASP_blkoi_data, 63
 - precond_block_data, 50
 - precond_block_reservoir_data, 53
 - precond_sweeping_data, 65
- REAL
 - fasp.h, 156
- RR
 - precond_FASP_blkoi_data, 63
 - precond_block_reservoir_data, 53
- RS_C1
 - fasp.h, 156
- rap.c, 352
 - fasp_blas_dcsr_rap2, 353
- restart
 - input_param, 44
 - itsolver_param, 46
 - precond_FASP_blkoi_data, 63
- s
 - grid2d, 34
- SA_AMG
 - fasp_const.h, 175
- SCHWARZ_BACKWARD
 - fasp_const.h, 175
- SCHWARZ_FORWARD
 - fasp_const.h, 175
- SCHWARZ_SYMMETRIC
 - fasp_const.h, 175
- SHORT
 - fasp.h, 157
- SMALLREAL
 - fasp_const.h, 176
- SMALLREAL2
 - fasp_const.h, 176
- SMOOTHER_BLKOil
 - fasp_const.h, 176
- SMOOTHER_CG
 - fasp_const.h, 176
- SMOOTHER_GS
 - fasp_const.h, 176
- SMOOTHER_GSOR
 - fasp_const.h, 176
- SMOOTHER_JACOBI
 - fasp_const.h, 176
- SMOOTHER_L1DIAG
 - fasp_const.h, 176
- SMOOTHER_POLY
 - fasp_const.h, 177
- SMOOTHER_SGS
 - fasp_const.h, 177
- SMOOTHER_SGSOR
 - fasp_const.h, 177
- SMOOTHER_SOR
 - fasp_const.h, 177
- SMOOTHER_SPETEN
 - fasp_const.h, 177
- SMOOTHER_SSOR
 - fasp_const.h, 177
- SOLVER_AMG
 - fasp_const.h, 177
- SOLVER_BiCGstab
 - fasp_const.h, 177
- SOLVER_CG
 - fasp_const.h, 177
- SOLVER_DEFAULT
 - fasp_const.h, 178
- SOLVER_FMG
 - fasp_const.h, 178
- SOLVER_GCG
 - fasp_const.h, 178
- SOLVER_GCR
 - fasp_const.h, 178
- SOLVER_GMRES
 - fasp_const.h, 178
- SOLVER_MUMPS
 - fasp_const.h, 178
- SOLVER_MinRes
 - fasp_const.h, 178
- SOLVER_SBiCGstab
 - fasp_const.h, 178

- SOLVER_SCG
 - fasp_const.h, [179](#)
- SOLVER_SGCG
 - fasp_const.h, [179](#)
- SOLVER_SGMRES
 - fasp_const.h, [179](#)
- SOLVER_SMinRes
 - fasp_const.h, [179](#)
- SOLVER_SUPERLU
 - fasp_const.h, [179](#)
- SOLVER_SVFGMRES
 - fasp_const.h, [179](#)
- SOLVER_SVGMRES
 - fasp_const.h, [179](#)
- SOLVER_UMFPACK
 - fasp_const.h, [179](#)
- SOLVER_VFGMRES
 - fasp_const.h, [179](#)
- SOLVER_VGMRES
 - fasp_const.h, [180](#)
- SS
 - precond_FASP_blkoi_data, [63](#)
 - precond_block_reservoir_data, [53](#)
- STAG_RATIO
 - fasp_const.h, [180](#)
- STOP_MOD_REL_RES
 - fasp_const.h, [180](#)
- STOP_REL_PRECRES
 - fasp_const.h, [180](#)
- STOP_REL_RES
 - fasp_const.h, [180](#)
- SWAP
 - smat.c, [357](#)
- scaled
 - precond_FASP_blkoi_data, [63](#)
 - precond_block_reservoir_data, [53](#)
- Schwarz_blk solver
 - input_param, [44](#)
- Schwarz_data, [66](#)
- Schwarz_maxlvl
 - input_param, [44](#)
- Schwarz_mmsize
 - input_param, [44](#)
- Schwarz_param, [67](#)
- schwarz_setup.c, [353](#)
 - fasp_Schwarz_get_block_matrix, [355](#)
 - fasp_Schwarz_setup, [355](#)
 - fasp_dcsr_Schwarz_backward_smoother, [354](#)
 - fasp_dcsr_Schwarz_forward_smoother, [354](#)
- Schwarz_type
 - input_param, [44](#)
- smat.c, [356](#)
 - fasp_blas_smat_Linfinity, [360](#)
 - fasp_blas_smat_inv, [357](#)
 - fasp_blas_smat_inv_nc, [357](#)
 - fasp_blas_smat_inv_nc2, [358](#)
 - fasp_blas_smat_inv_nc3, [358](#)
 - fasp_blas_smat_inv_nc4, [358](#)
 - fasp_blas_smat_inv_nc5, [359](#)
 - fasp_blas_smat_inv_nc7, [359](#)
 - fasp_blas_smat_invp_nc, [359](#)
 - fasp_iden_free, [360](#)
 - fasp_smat_identity, [360](#)
 - fasp_smat_identity_nc2, [362](#)
 - fasp_smat_identity_nc3, [362](#)
 - fasp_smat_identity_nc5, [362](#)
 - fasp_smat_identity_nc7, [363](#)
- SWAP, [357](#)
- smoother_bsr.c, [363](#)
 - fasp_smoother_dbsr_gs, [364](#)
 - fasp_smoother_dbsr_gs1, [365](#)
 - fasp_smoother_dbsr_gs_ascend, [365](#)
 - fasp_smoother_dbsr_gs_ascend1, [366](#)
 - fasp_smoother_dbsr_gs_descend, [366](#)
 - fasp_smoother_dbsr_gs_descend1, [367](#)
 - fasp_smoother_dbsr_gs_order1, [367](#)
 - fasp_smoother_dbsr_gs_order2, [368](#)
 - fasp_smoother_dbsr_ilu, [368](#)
 - fasp_smoother_dbsr_jacobi, [369](#)
 - fasp_smoother_dbsr_jacobi1, [369](#)
 - fasp_smoother_dbsr_jacobi_setup, [370](#)
 - fasp_smoother_dbsr_sor, [370](#)
 - fasp_smoother_dbsr_sor1, [371](#)
 - fasp_smoother_dbsr_sor_ascend, [371](#)
 - fasp_smoother_dbsr_sor_descend, [372](#)
 - fasp_smoother_dbsr_sor_order, [372](#)
- smoother_csr.c, [373](#)
 - fasp_smoother_dcsr_L1diag, [377](#)
 - fasp_smoother_dcsr_gs, [374](#)
 - fasp_smoother_dcsr_gs_cf, [374](#)
 - fasp_smoother_dcsr_gs_rb3d, [375](#)
 - fasp_smoother_dcsr_ilu, [375](#)
 - fasp_smoother_dcsr_jacobi, [376](#)
 - fasp_smoother_dcsr_kaczmarz, [376](#)
 - fasp_smoother_dcsr_sgs, [377](#)
 - fasp_smoother_dcsr_sor, [378](#)
 - fasp_smoother_dcsr_sor_cf, [378](#)
- smoother_csr_cr.c, [379](#)
 - fasp_smoother_dcsr_gscr, [379](#)
- smoother_csr_poly.c, [380](#)
 - fasp_smoother_dcsr_poly, [381](#)
 - fasp_smoother_dcsr_poly_old, [381](#)
- smoother_str.c, [381](#)
 - fasp_generate_diaginv_block, [383](#)
 - fasp_smoother_dstr_gs, [383](#)
 - fasp_smoother_dstr_gs1, [383](#)
 - fasp_smoother_dstr_gs_ascend, [384](#)
 - fasp_smoother_dstr_gs_cf, [384](#)

- fasp_smoother_dstr_gs_descend, 385
 - fasp_smoother_dstr_gs_order, 385
 - fasp_smoother_dstr_jacobi, 386
 - fasp_smoother_dstr_jacobi1, 386
 - fasp_smoother_dstr_schwarz, 387
 - fasp_smoother_dstr_sor, 387
 - fasp_smoother_dstr_sor1, 388
 - fasp_smoother_dstr_sor_ascend, 388
 - fasp_smoother_dstr_sor_cf, 389
 - fasp_smoother_dstr_sor_descend, 389
 - fasp_smoother_dstr_sor_order, 390
- solver_type
 - input_param, 44
- sparse_block.c, 390
 - fasp_bdcsr_free, 391
 - fasp_dbsr_Linfinity_dcsr, 392
 - fasp_dbsr_getblk, 391
 - fasp_dbsr_getblk_dcsr, 392
 - fasp_dcsr_getblk, 393
- sparse_bsr.c, 393
 - fasp_dbsr_alloc, 394
 - fasp_dbsr_cp, 395
 - fasp_dbsr_create, 395
 - fasp_dbsr_diagLU, 398
 - fasp_dbsr_diagLU2, 398
 - fasp_dbsr_diaginv, 396
 - fasp_dbsr_diaginv2, 396
 - fasp_dbsr_diaginv3, 397
 - fasp_dbsr_diaginv4, 397
 - fasp_dbsr_diagpref, 399
 - fasp_dbsr_free, 399
 - fasp_dbsr_getdiag, 400
 - fasp_dbsr_getdiaginv, 400
 - fasp_dbsr_null, 401
 - fasp_dbsr_trans, 401
- sparse_coo.c, 401
 - fasp_dcoo_alloc, 402
 - fasp_dcoo_create, 402
 - fasp_dcoo_free, 403
 - fasp_dcoo_shift, 403
- sparse_csr.c, 404
 - fasp_dcsr_alloc, 405
 - fasp_dcsr_compress, 405
 - fasp_dcsr_compress_inplace, 406
 - fasp_dcsr_cp, 406
 - fasp_dcsr_create, 407
 - fasp_dcsr_diagpref, 407
 - fasp_dcsr_free, 408
 - fasp_dcsr_getcol, 408
 - fasp_dcsr_getdiag, 408
 - fasp_dcsr_multicoloring, 409
 - fasp_dcsr_null, 409
 - fasp_dcsr_perm, 409
 - fasp_dcsr_regdiag, 411
 - fasp_dcsr_shift, 411
 - fasp_dcsr_sort, 412
 - fasp_dcsr_symdiagscale, 412
 - fasp_dcsr_sympat, 412
 - fasp_dcsr_trans, 414
 - fasp_icsr_cp, 414
 - fasp_icsr_create, 414
 - fasp_icsr_free, 416
 - fasp_icsr_null, 416
 - fasp_icsr_trans, 416
- sparse_csr.c, 418
 - fasp_dcscr_create, 418
 - fasp_dcscr_free, 419
- sparse_str.c, 419
 - fasp_dstr_alloc, 420
 - fasp_dstr_cp, 420
 - fasp_dstr_create, 421
 - fasp_dstr_free, 421
 - fasp_dstr_null, 421
- sparse_util.c, 422
 - fasp_sparse_MIS, 425
 - fasp_sparse_aat_, 423
 - fasp_sparse_abyb_, 423
 - fasp_sparse_abybms_, 424
 - fasp_sparse_aplbms_, 424
 - fasp_sparse_aplusb_, 425
 - fasp_sparse_iit_, 425
 - fasp_sparse_rapcmp_, 426
 - fasp_sparse_rapms_, 426
 - fasp_sparse_wta_, 427
 - fasp_sparse_wtams_, 428
 - fasp_sparse_ytx_, 428
 - fasp_sparse_ytxbig_, 428
- spbcgs.c, 429
 - fasp_solver_bdcsr_spbcgs, 430
 - fasp_solver_dbsr_spbcgs, 431
 - fasp_solver_dcsr_spbcgs, 431
 - fasp_solver_dstr_spbcgs, 433
- spcg.c, 434
 - fasp_solver_bdcsr_spcg, 435
 - fasp_solver_dcsr_spcg, 436
 - fasp_solver_dstr_spcg, 436
- spgmres.c, 437
 - fasp_solver_bdcsr_spgmres, 438
 - fasp_solver_dbsr_spgmres, 438
 - fasp_solver_dcsr_spgmres, 439
 - fasp_solver_dstr_spgmres, 439
- spminres.c, 440
 - fasp_solver_bdcsr_spminres, 442
 - fasp_solver_dcsr_spminres, 442
 - fasp_solver_dstr_spminres, 443
- spvgmres.c, 444
 - fasp_solver_bdcsr_spvgmres, 444
 - fasp_solver_dbsr_spvgmres, 445

- fasp_solver_dcsr_spvgmres, [445](#)
 - fasp_solver_dstr_spvgmres, [447](#)
- stop_type
 - input_param, [45](#)
 - itsolver_param, [46](#)
- t
 - grid2d, [34](#)
- THDs_AMG_GS
 - threads.c, [450](#)
- THDs_CPR_gGS
 - threads.c, [450](#)
- THDs_CPR_IGS
 - threads.c, [450](#)
- TRUE
 - fasp_const.h, [180](#)
- tfather
 - grid2d, [34](#)
- threads.c, [448](#)
 - FASP_GET_START_END, [448](#)
 - fasp_set_GS_threads, [448](#)
 - THDs_AMG_GS, [450](#)
 - THDs_CPR_gGS, [450](#)
 - THDs_CPR_IGS, [450](#)
- timing.c, [450](#)
 - fasp_gettime, [451](#)
- tol
 - itsolver_param, [46](#)
 - precond_FASP_blkoi_data, [63](#)
- total_alloc_count
 - fasp.h, [159](#)
 - memory.c, [267](#)
- total_alloc_mem
 - fasp.h, [159](#)
 - memory.c, [267](#)
- triangles
 - grid2d, [34](#)
- UA_AMG
 - fasp_const.h, [180](#)
- UNPT
 - fasp_const.h, [180](#)
- USERDEFINED
 - fasp_const.h, [181](#)
- V_CYCLE
 - fasp_const.h, [181](#)
- VMB
 - fasp_const.h, [181](#)
- val
 - dBSRmat, [28](#)
- vec.c, [451](#)
 - fasp_dvec_alloc, [452](#)
 - fasp_dvec_cp, [452](#)
 - fasp_dvec_create, [453](#)
 - fasp_dvec_free, [453](#)
 - fasp_dvec_isnan, [453](#)
 - fasp_dvec_maxdiff, [455](#)
 - fasp_dvec_null, [455](#)
 - fasp_dvec_rand, [456](#)
 - fasp_dvec_set, [456](#)
 - fasp_dvec_symdiagscale, [457](#)
 - fasp_ivec_alloc, [457](#)
 - fasp_ivec_create, [457](#)
 - fasp_ivec_free, [459](#)
 - fasp_ivec_set, [459](#)
- vertices
 - grid2d, [34](#)
- w
 - precond_FASP_blkoi_data, [63](#)
 - precond_block_reservoir_data, [53](#)
 - precond_sweeping_data, [65](#)
- W_CYCLE
 - fasp_const.h, [181](#)
- WW
 - precond_FASP_blkoi_data, [64](#)
 - precond_block_reservoir_data, [54](#)
- workdir
 - input_param, [45](#)
- wrapper.c, [460](#)
 - fasp_fwrapper_amg_, [461](#)
 - fasp_fwrapper_krylov_amg_, [462](#)
 - fasp_wrapper_dbsr_krylov_amg, [462](#)
 - fasp_wrapper_dcoo_dbsr_krylov_amg, [463](#)