FASP for Navier-Stokes

0.2.0 March/18/2018

Generated by Doxygen 1.8.14

Contents

Chapter 1

Data Structure Index

1.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 2

File Index

2.1 File List

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

Chapter 3

Data Structure Documentation

3.1 AMG_ns_data Struct Reference

Data for AMG solvers for Navier-Stokes problems.

`#include <fasp4ns.h>`

Data Fields

- SHORT max_levels
  max number of levels
- SHORT num_levels
  number of levels in use <= max_levels
- dBLCmat A
  pointer to the matrix at level level_num
- dBLCmat R
  restriction operator at level level_num
- dBLCmat 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
- INT cycle_type
  cycle type
- void * Numeric
  pointer to the numerical factorization from UMFPACK
- Pardiso_data pdata
  data for Intel MKL PARDISO
- Mumps_data mumps
  data for MUMPS
- dvector w
  Temporary work space.
- INT near_kernel_dim_v
  SA information.

- INT near_kernel_dim_p
  dimension of the near kernel for SAMG for pressure
- REAL ** near_kernel_basis_v
  basis of near kernel space for SAMG for velocity
- REAL ** near_kernel_basis_p
  basis of near kernel space for SAMG for pressure
- ivector cfmark_v
  pointer to the CF marker for velocity at level level_num
- ivector cfmark_p
  pointer to the CF marker for pressure at level level_num
- INT ILU_levels_v
  number of levels use ILU smoother for velocity
- INT ILU_levels_p
  number of levels use ILU smoother for pressure
- ILU_data LU_v
  ILU matrix for ILU smoother for velocity.
- ILU_data LU_p
  ILU matrix for ILU smoother for pressure.
- INT SWZ_levels_v
  number of levels use Schwarz smoother for velocity
- INT SWZ_levels_p
  number of levels use Schwarz smoother for pressure
- SWZ_data SWZ_v
  data of Schwarz smoother for velocity
- SWZ_data SWZ_p
  data of Schwarz smoother for pressure

3.1.1 Detailed Description

Data for AMG solvers for Navier-Stokes problems.

Note

This is needed for the AMG solver/preconditioner for Navier-Stokes problems

Definition at line 64 of file fasp4ns.h.

3.1.2 Field Documentation

3.1.2.1 near_kernel_dim_v

```
INT near_kernel_dim_v
```

SA information.

dimension of the near kernel for SAMG for velocity

Definition at line 113 of file fasp4ns.h.

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

- fasp4ns.h

3.2 AMG_param Struct Reference

Parameters for AMG solver.

`#include <fasp4ns.h>`

3.2.1 Detailed Description

Parameters for AMG solver.

Note

This is needed for the AMG solver/preconditioner.

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

- fasp4ns.h

3.3 input_param Struct Reference

Input parameters.

`#include <fasp4ns.h>`

3.3.1 Detailed Description

Input parameters.

Input parameters, reading from disk file

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

- fasp4ns.h

3.4 itsolver_ns_param Struct Reference

Parameters passed to iterative solvers.

`#include <fasp4ns.h>`

Data Fields

- SHORT itsolver_type
- SHORT precond_type
- SHORT stop_type
- INT maxit
- REAL tol
- INT restart
- SHORT print_level
- SHORT itsolver_type_v
- SHORT precond_type_v
- INT pre_maxit_v
- REAL pre_tol_v
- INT pre_restart_v
- SHORT print_level_v
- SHORT itsolver_type_p
- SHORT precond_type_p
- INT pre_maxit_p
- REAL pre_tol_p
- INT pre_restart_p
- SHORT print_level_p

3.4.1 Detailed Description

Parameters passed to iterative solvers.

Definition at line 164 of file fasp4ns.h.

3.4.2 Field Documentation

3.4.2.1 itsolver_type

```
SHORT itsolver_type
```

solver type: see message.h

Definition at line 169 of file fasp4ns.h.

3.4.2.2 itsolver_type_p

```
SHORT itsolver_type_p
```

solver type: see message.h

Definition at line 190 of file fasp4ns.h.

3.4.2.3 itsolver_type_v

```
SHORT itsolver_type_v
```

solver type: see message.h

Definition at line 180 of file fasp4ns.h.

3.4.2.4 maxit

```
INT maxit
```

max number of iterations

Definition at line 172 of file fasp4ns.h.

3.4.2.5 pre_maxit_p

```
INT pre_maxit_p
```

max number of iterations

Definition at line 192 of file fasp4ns.h.

3.4.2.6 pre_maxit_v

```
INT pre_maxit_v
```

max number of iterations

Definition at line 182 of file fasp4ns.h.

3.4.2.7 pre_restart_p

```
INT pre_restart_p
```

number of steps for restarting: for GMRES etc

Definition at line 194 of file fasp4ns.h.

3.4.2.8 pre_restart_v

```
INT pre_restart_v
```

number of steps for restarting: for GMRES etc

Definition at line 184 of file fasp4ns.h.

3.4.2.9 pre_tol_p

```
REAL pre_tol_p
```

convergence tolerance

Definition at line 193 of file fasp4ns.h.

3.4.2.10 pre_tol_v

```
REAL pre_tol_v
```

convergence tolerance

Definition at line 183 of file fasp4ns.h.

3.4.2.11 precond_type

```
SHORT precond_type
```

preconditioner type: see message.h

Definition at line 170 of file fasp4ns.h.

3.4.2.12 precond_type_p

```
SHORT precond_type_p
```

preconditioner type: see message.h

Definition at line 191 of file fasp4ns.h.

3.4.2.13 precond_type_v

```
SHORT precond_type_v
```

preconditioner type: see message.h

Definition at line 181 of file fasp4ns.h.

3.4.2.14 print_level

```
SHORT print_level
```

print level: 0–10

Definition at line 175 of file fasp4ns.h.

3.4.2.15 print_level_p

```
SHORT print_level_p
```

print level: 0–10

Definition at line 195 of file fasp4ns.h.

3.4.2.16 print_level_v

```
SHORT print_level_v
```

print level: 0–10

Definition at line 185 of file fasp4ns.h.

3.4.2.17 restart

```
INT restart
```

number of steps for restarting: for GMRES etc

Definition at line 174 of file fasp4ns.h.

3.4.2.18 stop_type

```
SHORT stop_type
```

stopping criteria type

Definition at line 171 of file fasp4ns.h.

3.4.2.19 tol

```
REAL tol
```

convergence tolerance

Definition at line 173 of file fasp4ns.h.

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

- fasp4ns.h

3.5 precond_ns_data Struct Reference

Data passed to the preconditioner for generalized Navier-Stokes problems.

```
#include <fasp4ns.h>
```

Data Fields

- int colA
  size of A, B, and whole matrix
- AMG_data * mgl_data_v
- int print_level
  print level in AMG preconditioner
- int max_levels
  max number of AMG levels
- int maxit
  max number of iterations of AMG preconditioner
- double amg_tol
  tolerance for AMG preconditioner
- int cycle_type
  AMG cycle type.
- int smoother
  AMG smoother type.
- int presmooth_iter
  number of presmoothing
- int postsmooth_iter
  number of postsmoothing

- int coarsening_type
  coarsening type
- double relaxation
  relaxation parameter for SOR smoother
- int coarse_scaling
  switch of scaling of coarse grid correction
- ILU_data * ILU_p
- dCSRmat * M
- dvector * diag_M
- dvector * diag_A
- dCSRmat * B
- dCSRmat * Bt
- dCSRmat * C
- dCSRmat * S
- dCSRmat * BABt
- dvector * diag_S
- dCSRmat * P
- dvector * rp
- dvector * sp
- ILU_data * LU_S
- double * w
  temporary work space

3.5.1 Detailed Description

Data passed to the preconditioner for generalized Navier-Stokes problems.

Definition at line 220 of file fasp4ns.h.

3.5.2 Field Documentation

3.5.2.1 B

`dCSRmat* B`

matrix B

Definition at line 301 of file fasp4ns.h.

3.5.2.2 BABt

`dCSRmat* BABt`

matrix BABt

Definition at line 305 of file fasp4ns.h.

3.5.2.3 Bt

```
dCSRmat* Bt
```

matrix of transpose of B

Definition at line 302 of file fasp4ns.h.

3.5.2.4 C

```
dCSRmat* C
```

matrix C

Definition at line 303 of file fasp4ns.h.

3.5.2.5 diag_A

```
dvector* diag_A
```

diagonal of velocity block A

Definition at line 300 of file fasp4ns.h.

3.5.2.6 diag_M

```
dvector* diag_M
```

diagonal of mass matrix M

Definition at line 299 of file fasp4ns.h.

3.5.2.7 diag_S

```
dvector* diag_S
```

diagonal of Schur Complement matrix S

Definition at line 306 of file fasp4ns.h.

3.5.2.8 ILU_p

```
ILU_data* ILU_p
```

ILU data for presure block

Definition at line 293 of file fasp4ns.h.

3.5.2.9 LU_S

```
ILU_data* LU_S
```

LU date for schur

Definition at line 310 of file fasp4ns.h.

3.5.2.10 M

```
dCSRmat* M
```

mass matrix for pressure

Definition at line 298 of file fasp4ns.h.

3.5.2.11 mgl_data_v

```
AMG_data* mgl_data_v
```

what is this? – Xiaozhe

Definition at line 230 of file fasp4ns.h.

3.5.2.12 P

```
dCSRmat* P
```

Poisson matrix of pressure

Definition at line 307 of file fasp4ns.h.

3.5.2.13 rp

```
dvector* rp
```

residual for pressure

Definition at line 308 of file fasp4ns.h.

3.5.2.14 S

```
dCSRmat* S
```

Schur Complement matrix

Definition at line 304 of file fasp4ns.h.

3.5.2.15 sp

```
dvector* sp
```

sol for pressure

Definition at line 309 of file fasp4ns.h.

3.5.2.16 w

```
double* w
```

temporary work space

temporary work space for other usage

Definition at line 314 of file fasp4ns.h.

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

- fasp4ns.h

3.6 precond_ns_param Struct Reference

Parameters passed to the preconditioner for generalized Navier-Stokes problems.

```
#include <fasp4ns.h>
```

Data Fields

- int AMG_type
  AMG type.
- int print_level
  print level in AMG preconditioner
- int max_levels
  max number of AMG levels

3.6.1 Detailed Description

Parameters passed to the preconditioner for generalized Navier-Stokes problems.

Definition at line 204 of file fasp4ns.h.

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

- fasp4ns.h

3.7 precond_pnp_stokes_data Struct Reference

Data passed to the preconditioner for block preconditioning for dBLCmat format.

`#include <fasp4ns.h>`

Data Fields

- dBLCmat * Abcsr
- dCSRmat * A_pnp_csr
- dBSRmat * A_pnp_bsr
- dCSRmat * A_stokes_csr
- dBLCmat * A_stokes_bcsr
- dvector r
- void ** LU_diag
- precond_data_bsr * precdata_pnp
- precond_ns_data * precdata_stokes

3.7.1 Detailed Description

Data passed to the preconditioner for block preconditioning for dBLCmat format.

This is needed for the block preconditioner for pnp+stokes system.

Definition at line 472 of file fasp4ns.h.

3.7.2 Field Documentation

3.7.2.1 A_pnp_bsr

```
dBSRmat* A_pnp_bsr
```

data for pnp diagonal block in bsr format

Definition at line 480 of file fasp4ns.h.

3.7.2.2 A_pnp_csr

```
dCSRmat* A_pnp_csr
```

data for pnp diagonal block in csr format

Definition at line 479 of file fasp4ns.h.

3.7.2.3 A_stokes_bcsr

```
dBLCmat* A_stokes_bcsr
```

data for pnp diagonal block in bsr format

Definition at line 483 of file fasp4ns.h.

3.7.2.4 A_stokes_csr

```
dCSRmat* A_stokes_csr
```

data for pnp diagonal block in csr format

Definition at line 482 of file fasp4ns.h.

3.7.2.5 Abcsr

```
dBLCmat* Abcsr
```

problem data, the blocks

Definition at line 477 of file fasp4ns.h.

3.7.2.6 LU_diag

```
void** LU_diag
```

LU decomposition for the diagonal blocks (for UMFpack)

Definition at line 491 of file fasp4ns.h.

3.7.2.7 precdata_pnp

```
precond_data_bsr* precdata_pnp
```

data for pnp diagonal block

Definition at line 496 of file fasp4ns.h.

3.7.2.8 precdata_stokes

```
precond_ns_data* precdata_stokes
```

data for stokes diagonal block

Definition at line 499 of file fasp4ns.h.

3.7.2.9 r

```
dvector r
```

temp work space

Definition at line 485 of file fasp4ns.h.

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

- fasp4ns.h

Chapter 4

File Documentation

4.1 assemble_util.inl File Reference

Subroutines for assembling purpose.

4.1.1 Detailed Description

Subroutines for assembling purpose.

4.2 AuxInput.c File Reference

Read and check input parameters.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- SHORT fasp_ns_param_check (const input_ns_param *inparam)

  Simple check on input parameters.
- void fasp_ns_param_input (char *filenm, input_ns_param *Input)

  Read input parameters for NS problem from disk file.

4.2.1 Detailed Description

Read and check input parameters.

Note

This file contains Level-0 (Aux) functions.

4.2.2 Function Documentation

4.2.2.1 fasp_ns_param_check()

```
SHORT fasp_ns_param_check (
            const input_ns_param * inparam )
```

Simple check on input parameters.

Parameters

inparam	Input parameters

Author

Chensong Zhang

Date

09/29/2013

Modified by Xiaozhe Hu on 05/27/2014 Modified by Chensong Zhang on 03/18/2018

Definition at line 36 of file AuxInput.c.

```
37 {
38     SHORT status = FASP_SUCCESS;
39
40     if ( inparam->problem_num<0
41          || inparam->solver_type<0
42          || inparam->solver_type>50
43          || inparam->precond_type<0
44          || inparam->itsolver_tol<=0
45          || inparam->itsolver_maxit<=0
46          || inparam->stop_type<=0
47          || inparam->stop_type>3
48          || inparam->restart<0
49          || inparam->ILU_type<=0
50          || inparam->ILU_type>3
51          || inparam->ILU_lfil<0
52          || inparam->ILU_droptol<=0
53          || inparam->ILU_relax<0
54          || inparam->ILU_permtol<0
```

```
55          || inparam->SWZ_mmsize<0
56          || inparam->SWZ_maxlvl<0
57          || inparam->SWZ_type<0
58          || inparam->AMG_type_v<=0
59          || inparam->AMG_type_v>3
60          || inparam->AMG_cycle_type_v<=0
61          || inparam->AMG_cycle_type_v>4
62          || inparam->AMG_levels_v<0
63          || inparam->AMG_ILU_levels_v<0
64          || inparam->AMG_coarse_dof_v<=0
65          || inparam->AMG_tol_v<0
66          || inparam->AMG_maxit_v<0
67          || inparam->AMG_coarsening_type_v<=0
68          || inparam->AMG_coarsening_type_v>4
69          || inparam->AMG_interpolation_type_v<0
70          || inparam->AMG_interpolation_type_v>5
71          || inparam->AMG_smoother_v<0
72          || inparam->AMG_smoother_v>20
73          || inparam->AMG_strong_threshold_v<0.0
74          || inparam->AMG_strong_threshold_v>0.9999
75          || inparam->AMG_truncation_threshold_v<0.0
76          || inparam->AMG_truncation_threshold_v>0.9999
77          || inparam->AMG_max_row_sum_v<0.0
78          || inparam->AMG_presmooth_iter_v<0
79          || inparam->AMG_postsmooth_iter_v<0
80          || inparam->AMG_amli_degree_v<0
81          || inparam->AMG_aggressive_level_v<0
82          || inparam->AMG_aggressive_path_v<0
83          || inparam->AMG_strong_coupled_v<0
84          || inparam->AMG_max_aggregation_v<=0
85          || inparam->AMG_tentative_smooth_v<0
86          || inparam->AMG_smooth_filter_v<0
87          || inparam->AMG_type_p<=0
88          || inparam->AMG_type_p>3
89          || inparam->AMG_cycle_type_p<=0
90          || inparam->AMG_cycle_type_p>4
91          || inparam->AMG_levels_p<0
92          || inparam->AMG_ILU_levels_p<0
93          || inparam->AMG_coarse_dof_p<=0
94          || inparam->AMG_tol_p<0
95          || inparam->AMG_maxit_p<0
96          || inparam->AMG_coarsening_type_p<=0
97          || inparam->AMG_coarsening_type_p>4
98          || inparam->AMG_interpolation_type_p<0
99          || inparam->AMG_interpolation_type_p>5
100          || inparam->AMG_smoother_p<0
101          || inparam->AMG_smoother_p>20
102          || inparam->AMG_strong_threshold_p<0.0
103          || inparam->AMG_strong_threshold_p>0.9999
104          || inparam->AMG_truncation_threshold_p<0.0
105          || inparam->AMG_truncation_threshold_p>0.9999
106          || inparam->AMG_max_row_sum_p<0.0
107          || inparam->AMG_presmooth_iter_p<0
108          || inparam->AMG_postsmooth_iter_p<0
109          || inparam->AMG_amli_degree_p<0
110          || inparam->AMG_aggressive_level_p<0
111          || inparam->AMG_aggressive_path_p<0
112          || inparam->AMG_strong_coupled_p<0
113          || inparam->AMG_max_aggregation_p<=0
114          || inparam->AMG_tentative_smooth_p<0
115          || inparam->AMG_smooth_filter_p<0
116         ) status = ERROR_INPUT_PAR;
117
118     return status;
119 }
```

4.2.2.2 fasp_ns_param_input()

```
void fasp_ns_param_input (
             char * filenm,
             input_ns_param * Input )
```

Read input parameters for NS problem from disk file.

Parameters

filenm	File name for input file
Input	Input parameters

Author

Lu Wang

Date

02/15/2012

Modified by Chensong Zhang on 03/27/2017: check unexpected error Modified by Chensong Zhang on 09/23/2017: new skip the line Modified by Chensong Zhang on 03/18/2018: format

Definition at line 136 of file AuxInput.c.

```
138 {
139     char     buffer[500]; // Note: max number of char for each line!
140     INT      val;
141     SHORT    status = FASP_SUCCESS;
142
143     // set default input parameters
144     fasp_ns_param_input_init(Input);
145
146     // if input file is not specified, use the default values
147     if (filenm==NULL) return;
148
149     FILE *fp = fopen(filenm,"r");
150     if (fp==NULL) {
151         printf("### ERROR: Could not open file %s...\n", filenm);
152         exit(ERROR_OPEN_FILE);
153     }
154
155     while ( status == FASP_SUCCESS ) {
156         INT   ibuff;
157         REAL  dbuff;
158         char  sbuff[500];
159
160         val = fscanf(fp,"%s",buffer);
161
162         if (val==EOF) break;
163         if (val!=1) { status = ERROR_INPUT_PAR; break; }
164         if (buffer[0]=='[' || buffer[0]=='%' || buffer[0]=='|') {
165             fscanf(fp, "%*[^\n]"); // skip rest of line
166             continue;
167         }
168
169         // match keyword and scan for value
170         if (strcmp(buffer,"workdir")==0) {
171             val = fscanf(fp,"%s",buffer);
172             if (val!=1 || strcmp(buffer,"=")!=0) {
173                 status = ERROR_INPUT_PAR; break;
174             }
175             val = fscanf(fp,"%s",sbuff);
176             if (val!=1) { status = ERROR_INPUT_PAR; break; }
177             strncpy(Input->workdir,sbuff,128);
178             fscanf(fp, "%*[^\n]"); // skip rest of line
179         }
180
181         else if (strcmp(buffer,"problem_num")==0) {
182             val = fscanf(fp,"%s",buffer);
183             if (val!=1 || strcmp(buffer,"=")!=0) {
184                 status = ERROR_INPUT_PAR; break;
185             }
186             val = fscanf(fp,"%d",&ibuff);
187             if (val!=1) { status = ERROR_INPUT_PAR; break; }
188             Input->problem_num=ibuff;
189             fscanf(fp, "%*[^\n]"); // skip rest of line
190         }
191
192         else if (strcmp(buffer,"print_level")==0) {
```

```
193             val = fscanf(fp,"%s",buffer);
194             if (val!=1 || strcmp(buffer,"=")!=0) {
195                 status = ERROR_INPUT_PAR; break;
196             }
197             val = fscanf(fp,"%d",&ibuff);
198             if (val!=1) { status = ERROR_INPUT_PAR; break; }
199             Input->print_level = ibuff;
200             fscanf(fp, "%*[^\n]"); // skip rest of line
201         }
202 
203         else if (strcmp(buffer,"output_type")==0) {
204             val = fscanf(fp,"%s",buffer);
205             if (val!=1 || strcmp(buffer,"=")!=0) {
206                 status = ERROR_INPUT_PAR; break;
207             }
208             val = fscanf(fp,"%d",&ibuff);
209             if (val!=1) { status = ERROR_INPUT_PAR; break; }
210             Input->output_type = ibuff;
211             fscanf(fp, "%*[^\n]"); // skip rest of line
212         }
213 
214         else if (strcmp(buffer,"solver_type")==0) {
215             val = fscanf(fp,"%s",buffer);
216             if (val!=1 || strcmp(buffer,"=")!=0) {
217                 status = ERROR_INPUT_PAR; break;
218             }
219             val = fscanf(fp,"%d",&ibuff);
220             if (val!=1) { status = ERROR_INPUT_PAR; break; }
221             Input->solver_type = ibuff;
222             fscanf(fp, "%*[^\n]"); // skip rest of line
223         }
224 
225         else if (strcmp(buffer,"precond_type")==0) {
226             val = fscanf(fp,"%s",buffer);
227             if (val!=1 || strcmp(buffer,"=")!=0) {
228                 status = ERROR_INPUT_PAR; break;
229             }
230             val = fscanf(fp,"%d",&ibuff);
231             if (val!=1) { status = ERROR_INPUT_PAR; break; }
232             Input->precond_type = ibuff;
233             fscanf(fp, "%*[^\n]"); // skip rest of line
234         }
235 
236         else if (strcmp(buffer,"stop_type")==0) {
237             val = fscanf(fp,"%s",buffer);
238             if (val!=1 || strcmp(buffer,"=")!=0) {
239                 status = ERROR_INPUT_PAR; break;
240             }
241             val = fscanf(fp,"%d",&ibuff);
242             if (val!=1) { status = ERROR_INPUT_PAR; break; }
243             Input->stop_type = ibuff;
244             fscanf(fp, "%*[^\n]"); // skip rest of line
245         }
246 
247         else if (strcmp(buffer,"itsolver_tol")==0) {
248             val = fscanf(fp,"%s",buffer);
249             if (val!=1 || strcmp(buffer,"=")!=0) {
250                 status = ERROR_INPUT_PAR; break;
251             }
252             val = fscanf(fp,"%lf",&dbuff);
253             if (val!=1) { status = ERROR_INPUT_PAR; break; }
254             Input->itsolver_tol = dbuff;
255             fscanf(fp, "%*[^\n]"); // skip rest of line
256         }
257 
258         else if (strcmp(buffer,"itsolver_maxit")==0) {
259             val = fscanf(fp,"%s",buffer);
260             if (val!=1 || strcmp(buffer,"=")!=0) {
261                 status = ERROR_INPUT_PAR; break;
262             }
263             val = fscanf(fp,"%d",&ibuff);
264             if (val!=1) { status = ERROR_INPUT_PAR; break; }
265             Input->itsolver_maxit = ibuff;
266             fscanf(fp, "%*[^\n]"); // skip rest of line
267         }
268 
269         else if (strcmp(buffer,"solver_type_v")==0) {
270             val = fscanf(fp,"%s",buffer);
271             if (val!=1 || strcmp(buffer,"=")!=0) {
272                 status = ERROR_INPUT_PAR; break;
273             }
274             val = fscanf(fp,"%d",&ibuff);
275             if (val!=1) { status = ERROR_INPUT_PAR; break; }
276             Input->itsolver_type_v = ibuff;
277             fscanf(fp, "%*[^\n]"); // skip rest of line
278         }
279 
```

```
280         else if (strcmp(buffer,"precond_type_v")==0) {
281             val = fscanf(fp,"%s",buffer);
282             if (val!=1 || strcmp(buffer,"=")!=0) {
283                 status = ERROR_INPUT_PAR; break;
284             }
285             val = fscanf(fp,"%d",&ibuff);
286             if (val!=1) { status = ERROR_INPUT_PAR; break; }
287             Input->precond_type_v = ibuff;
288             fscanf(fp, "%*[^\n]"); // skip rest of line
289         }
290 
291         else if (strcmp(buffer,"itsolver_tol_v")==0) {
292             val = fscanf(fp,"%s",buffer);
293             if (val!=1 || strcmp(buffer,"=")!=0) {
294                 status = ERROR_INPUT_PAR; break;
295             }
296             val = fscanf(fp,"%lf",&dbuff);
297             if (val!=1) { status = ERROR_INPUT_PAR; break; }
298             Input->pre_tol_v = dbuff;
299             fscanf(fp, "%*[^\n]"); // skip rest of line
300         }
301 
302         else if (strcmp(buffer,"itsolver_maxit_v")==0) {
303             val = fscanf(fp,"%s",buffer);
304             if (val!=1 || strcmp(buffer,"=")!=0) {
305                 status = ERROR_INPUT_PAR; break;
306             }
307             val = fscanf(fp,"%d",&ibuff);
308             if (val!=1) { status = ERROR_INPUT_PAR; break; }
309             Input->pre_maxit_v = ibuff;
310             fscanf(fp, "%*[^\n]"); // skip rest of line
311         }
312 
313         else if (strcmp(buffer,"itsolver_restart_v")==0) {
314             val = fscanf(fp,"%s",buffer);
315             if (val!=1 || strcmp(buffer,"=")!=0) {
316                 status = ERROR_INPUT_PAR; break;
317             }
318             val = fscanf(fp,"%d",&ibuff);
319             if (val!=1) { status = ERROR_INPUT_PAR; break; }
320             Input->pre_restart_v = ibuff;
321             fscanf(fp, "%*[^\n]"); // skip rest of line
322         }
323 
324         else if (strcmp(buffer,"solver_type_p")==0) {
325             val = fscanf(fp,"%s",buffer);
326             if (val!=1 || strcmp(buffer,"=")!=0) {
327                 status = ERROR_INPUT_PAR; break;
328             }
329             val = fscanf(fp,"%d",&ibuff);
330             if (val!=1) { status = ERROR_INPUT_PAR; break; }
331             Input->itsolver_type_p = ibuff;
332             fscanf(fp, "%*[^\n]"); // skip rest of line
333         }
334 
335         else if (strcmp(buffer,"precond_type_p")==0) {
336             val = fscanf(fp,"%s",buffer);
337             if (val!=1 || strcmp(buffer,"=")!=0) {
338                 status = ERROR_INPUT_PAR; break;
339             }
340             val = fscanf(fp,"%d",&ibuff);
341             if (val!=1) { status = ERROR_INPUT_PAR; break; }
342             Input->precond_type_p = ibuff;
343             fscanf(fp, "%*[^\n]"); // skip rest of line
344         }
345 
346         else if (strcmp(buffer,"itsolver_tol_p")==0) {
347             val = fscanf(fp,"%s",buffer);
348             if (val!=1 || strcmp(buffer,"=")!=0) {
349                 status = ERROR_INPUT_PAR; break;
350             }
351             val = fscanf(fp,"%lf",&dbuff);
352             if (val!=1) { status = ERROR_INPUT_PAR; break; }
353             Input->pre_tol_p = dbuff;
354             fscanf(fp, "%*[^\n]"); // skip rest of line
355         }
356 
357         else if (strcmp(buffer,"itsolver_maxit_p")==0) {
358             val = fscanf(fp,"%s",buffer);
359             if (val!=1 || strcmp(buffer,"=")!=0) {
360                 status = ERROR_INPUT_PAR; break;
361             }
362             val = fscanf(fp,"%d",&ibuff);
363             if (val!=1) { status = ERROR_INPUT_PAR; break; }
364             Input->pre_maxit_p = ibuff;
365             fscanf(fp, "%*[^\n]"); // skip rest of line
366         }
```

```
367
368         else if (strcmp(buffer,"itsolver_restart_p")==0) {
369             val = fscanf(fp,"%s",buffer);
370             if (val!=1 || strcmp(buffer,"=")!=0) {
371                 status = ERROR_INPUT_PAR; break;
372             }
373             val = fscanf(fp,"%d",&ibuff);
374             if (val!=1) { status = ERROR_INPUT_PAR; break; }
375             Input->pre_restart_p = ibuff;
376             fscanf(fp, "%*[^\n]"); // skip rest of line
377         }
378
379         else if (strcmp(buffer,"itsolver_restart")==0) {
380             val = fscanf(fp,"%s",buffer);
381             if (val!=1 || strcmp(buffer,"=")!=0) {
382                 status = ERROR_INPUT_PAR; break;
383             }
384             val = fscanf(fp,"%d",&ibuff);
385             if (val!=1) { status = ERROR_INPUT_PAR; break; }
386             Input->restart = ibuff;
387             fscanf(fp, "%*[^\n]"); // skip rest of line
388         }
389
390         else if (strcmp(buffer,"AMG_ILU_levels_v")==0) {
391             val = fscanf(fp,"%s",buffer);
392             if (val!=1 || strcmp(buffer,"=")!=0) {
393                 status = ERROR_INPUT_PAR; break;
394             }
395             val = fscanf(fp,"%d",&ibuff);
396             if (val!=1) { status = ERROR_INPUT_PAR; break; }
397             Input->AMG_ILU_levels_v = ibuff;
398             fscanf(fp, "%*[^\n]"); // skip rest of line
399         }
400
401         else if (strcmp(buffer,"AMG_schwarz_levels_v")==0) {
402             val = fscanf(fp,"%s",buffer);
403             if (val!=1 || strcmp(buffer,"=")!=0) {
404                 status = ERROR_INPUT_PAR; break;
405             }
406             val = fscanf(fp,"%d",&ibuff);
407             if (val!=1) { status = FASP_SUCCESS; break; }
408             Input->AMG_schwarz_levels_v = ibuff;
409             fscanf(fp, "%*[^\n]"); // skip rest of line
410         }
411
412         else if (strcmp(buffer,"AMG_type_v")==0) {
413             val = fscanf(fp,"%s",buffer);
414             if (val!=1 || strcmp(buffer,"=")!=0) {
415                 status = ERROR_INPUT_PAR; break;
416             }
417             val = fscanf(fp,"%s",buffer);
418             if (val!=1) { status = ERROR_INPUT_PAR; break; }
419
420             if ((strcmp(buffer,"C")==0)||(strcmp(buffer,"c")==0))
421             Input->AMG_type_v = CLASSIC_AMG;
422             else if ((strcmp(buffer,"SA")==0)||(strcmp(buffer,"sa")==0))
423             Input->AMG_type_v = SA_AMG;
424             else if ((strcmp(buffer,"UA")==0)||(strcmp(buffer,"ua")==0))
425             Input->AMG_type_v = UA_AMG;
426             else
427             { status = ERROR_INPUT_PAR; break; }
428             fscanf(fp, "%*[^\n]"); // skip rest of line
429         }
430
431         else if (strcmp(buffer,"AMG_aggregation_type_v")==0) {
432             val = fscanf(fp,"%s",buffer);
433             if (val!=1 || strcmp(buffer,"=")!=0) {
434                 status = ERROR_INPUT_PAR; break;
435             }
436             val = fscanf(fp,"%d",&ibuff);
437             if (val!=1) { status = ERROR_INPUT_PAR; break; }
438             Input->AMG_aggregation_type_v = ibuff;
439             fscanf(fp, "%*[^\n]"); // skip rest of line
440         }
441
442         else if (strcmp(buffer,"AMG_pair_number_v")==0) {
443             val = fscanf(fp,"%s",buffer);
444             if (val!=1 || strcmp(buffer,"=")!=0) {
445                 status = ERROR_INPUT_PAR; break;
446             }
447             val = fscanf(fp,"%d",&ibuff);
448             if (val!=1) { status = ERROR_INPUT_PAR; break; }
449             Input->AMG_pair_number_v = ibuff;
450             fscanf(fp, "%*[^\n]"); // skip rest of line
451         }
452
453         else if (strcmp(buffer,"AMG_quality_bound_v")==0) {
```

```
454             val = fscanf(fp,"%s",buffer);
455             if (val!=1 || strcmp(buffer,"=")!=0) {
456                 status = ERROR_INPUT_PAR; break;
457             }
458             val = fscanf(fp,"%lf",&dbuff);
459             if (val!=1) { status = ERROR_INPUT_PAR; break; }
460             Input->AMG_quality_bound_v = dbuff;
461             fscanf(fp, "%*[^\n]"); // skip rest of line
462         }
463 
464         else if (strcmp(buffer,"AMG_strong_coupled_v")==0) {
465             val = fscanf(fp,"%s",buffer);
466             if (val!=1 || strcmp(buffer,"=")!=0) {
467                 status = ERROR_INPUT_PAR; break;
468             }
469             val = fscanf(fp,"%lf",&dbuff);
470             if (val!=1) { status = ERROR_INPUT_PAR; break; }
471             Input->AMG_strong_coupled_v = dbuff;
472             fscanf(fp, "%*[^\n]"); // skip rest of line
473         }
474 
475         else if (strcmp(buffer,"AMG_max_aggregation_v")==0) {
476             val = fscanf(fp,"%s",buffer);
477             if (val!=1 || strcmp(buffer,"=")!=0) {
478                 status = ERROR_INPUT_PAR; break;
479             }
480             val = fscanf(fp,"%d",&ibuff);
481             if (val!=1) { status = ERROR_INPUT_PAR; break; }
482             Input->AMG_max_aggregation_v = ibuff;
483             fscanf(fp, "%*[^\n]"); // skip rest of line
484         }
485 
486         else if (strcmp(buffer,"AMG_tentative_smooth_v")==0) {
487             val = fscanf(fp,"%s",buffer);
488             if (val!=1 || strcmp(buffer,"=")!=0) {
489                 status = ERROR_INPUT_PAR; break;
490             }
491             val = fscanf(fp,"%lf",&dbuff);
492             if (val!=1) { status = ERROR_INPUT_PAR; break; }
493             Input->AMG_tentative_smooth_v = dbuff;
494             fscanf(fp, "%*[^\n]"); // skip rest of line
495         }
496 
497         else if (strcmp(buffer,"AMG_smooth_filter_v")==0) {
498             val = fscanf(fp,"%s",buffer);
499             if (val!=1 || strcmp(buffer,"=")!=0) {
500                 status = ERROR_INPUT_PAR; break;
501             }
502             val = fscanf(fp,"%s",buffer);
503             if (val!=1) { status = ERROR_INPUT_PAR; break; }
504 
505             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
506                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
507             Input->AMG_smooth_filter_v = ON;
508             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
509                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
510                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
511                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
512             Input->AMG_smooth_filter_v = OFF;
513             else
514             { status = ERROR_INPUT_PAR; break; }
515             fscanf(fp, "%*[^\n]"); // skip rest of line
516         }
517 
518         else if (strcmp(buffer,"AMG_coarse_scaling_v")==0) {
519             val = fscanf(fp,"%s",buffer);
520             if (val!=1 || strcmp(buffer,"=")!=0) {
521                 status = ERROR_INPUT_PAR; break;
522             }
523             val = fscanf(fp,"%s",buffer);
524             if (val!=1) { status = ERROR_INPUT_PAR; break; }
525 
526             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
527                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
528             Input->AMG_coarse_scaling_v = ON;
529             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
530                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
531                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
532                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
533             Input->AMG_coarse_scaling_v = OFF;
534             else
535             { status = ERROR_INPUT_PAR; break; }
536             fscanf(fp, "%*[^\n]"); // skip rest of line
537         }
538 
539         else if (strcmp(buffer,"AMG_levels_v")==0) {
540             val = fscanf(fp,"%s",buffer);
```

```
541             if (val!=1 || strcmp(buffer,"=")!=0) {
542                 status = ERROR_INPUT_PAR; break;
543             }
544             val = fscanf(fp,"%d",&ibuff);
545             if (val!=1) { status = ERROR_INPUT_PAR; break; }
546             Input->AMG_levels_v = ibuff;
547             fscanf(fp, "%*[^\n]"); // skip rest of line
548         }
549 
550         else if (strcmp(buffer,"AMG_tol_v")==0) {
551             val = fscanf(fp,"%s",buffer);
552             if (val!=1 || strcmp(buffer,"=")!=0) {
553                 status = ERROR_INPUT_PAR; break;
554             }
555             val = fscanf(fp,"%lf",&dbuff);
556             if (val!=1) { status = ERROR_INPUT_PAR; break; }
557             Input->AMG_tol_v = dbuff;
558             fscanf(fp, "%*[^\n]"); // skip rest of line
559         }
560 
561         else if (strcmp(buffer,"AMG_maxit_v")==0) {
562             val = fscanf(fp,"%s",buffer);
563             if (val!=1 || strcmp(buffer,"=")!=0) {
564                 status = ERROR_INPUT_PAR; break;
565             }
566             val = fscanf(fp,"%d",&ibuff);
567             if (val!=1) { status = ERROR_INPUT_PAR; break; }
568             Input->AMG_maxit_v = ibuff;
569             fscanf(fp, "%*[^\n]"); // skip rest of line
570         }
571 
572         else if (strcmp(buffer,"AMG_coarse_dof_v")==0) {
573             val = fscanf(fp,"%s",buffer);
574             if (val!=1 || strcmp(buffer,"=")!=0) {
575                 status = ERROR_INPUT_PAR; break;
576             }
577             val = fscanf(fp,"%d",&ibuff);
578             if (val!=1) { status = ERROR_INPUT_PAR; break; }
579             Input->AMG_coarse_dof_v = ibuff;
580             fscanf(fp, "%*[^\n]"); // skip rest of line
581         }
582 
583         else if (strcmp(buffer,"AMG_coarse_solver_v")==0) {
584             val = fscanf(fp,"%s",buffer);
585             if (val!=1 || strcmp(buffer,"=")!=0) {
586                 status = ERROR_INPUT_PAR; break;
587             }
588             val = fscanf(fp,"%d",&ibuff);
589             if (val!=1) { status = ERROR_INPUT_PAR; break; }
590             Input->AMG_coarse_solver_v = ibuff;
591             fscanf(fp, "%*[^\n]"); // skip rest of line
592         }
593 
594         else if (strcmp(buffer,"AMG_cycle_type_v")==0) {
595             val = fscanf(fp,"%s",buffer);
596             if (val!=1 || strcmp(buffer,"=")!=0) {
597                 status = ERROR_INPUT_PAR; break;
598             }
599             val = fscanf(fp,"%s",buffer);
600             if (val!=1) { status = ERROR_INPUT_PAR; break; }
601 
602             if ((strcmp(buffer,"V")==0)||(strcmp(buffer,"v")==0))
603             Input->AMG_cycle_type_v = V_CYCLE;
604             else if ((strcmp(buffer,"W")==0)||(strcmp(buffer,"w")==0))
605             Input->AMG_cycle_type_v = W_CYCLE;
606             else if ((strcmp(buffer,"A")==0)||(strcmp(buffer,"a")==0))
607             Input->AMG_cycle_type_v = AMLI_CYCLE;
608             else if ((strcmp(buffer,"NA")==0)||(strcmp(buffer,"na")==0))
609             Input->AMG_cycle_type_v = NL_AMLI_CYCLE;
610             else
611             { status = ERROR_INPUT_PAR; break; }
612             fscanf(fp, "%*[^\n]"); // skip rest of line
613         }
614 
615         else if (strcmp(buffer,"AMG_smoother_v")==0) {
616             val = fscanf(fp,"%s",buffer);
617             if (val!=1 || strcmp(buffer,"=")!=0) {
618                 status = ERROR_INPUT_PAR; break;
619             }
620             val = fscanf(fp,"%s",buffer);
621             if (val!=1) { status = ERROR_INPUT_PAR; break; }
622 
623             if ((strcmp(buffer,"JACOBI")==0)||(strcmp(buffer,"jacobi")==0))
624             Input->AMG_smoother_v = SMOOTHER_JACOBI;
625             else if ((strcmp(buffer,"GS")==0)||(strcmp(buffer,"gs")==0))
626             Input->AMG_smoother_v = SMOOTHER_GS;
627             else if ((strcmp(buffer,"SGS")==0)||(strcmp(buffer,"sgs")==0))
```

```
628             Input->AMG_smoother_v = SMOOTHER_SGS;
629             else if ((strcmp(buffer,"CG")==0)||(strcmp(buffer,"cg")==0))
630             Input->AMG_smoother_v = SMOOTHER_CG;
631             else if ((strcmp(buffer,"SOR")==0)||(strcmp(buffer,"sor")==0))
632             Input->AMG_smoother_v = SMOOTHER_SOR;
633             else if ((strcmp(buffer,"SSOR")==0)||(strcmp(buffer,"ssor")==0))
634             Input->AMG_smoother_v = SMOOTHER_SSOR;
635             else if ((strcmp(buffer,"GSOR")==0)||(strcmp(buffer,"gsor")==0))
636             Input->AMG_smoother_v = SMOOTHER_GSOR;
637             else if ((strcmp(buffer,"SGSOR")==0)||(strcmp(buffer,"sgsor")==0))
638             Input->AMG_smoother_v = SMOOTHER_SGSOR;
639             else if ((strcmp(buffer,"POLY")==0)||(strcmp(buffer,"poly")==0))
640             Input->AMG_smoother_v = SMOOTHER_POLY;
641             else if ((strcmp(buffer,"L1_DIAG")==0)||(strcmp(buffer,"l1_diag")==0))
642             Input->AMG_smoother_v = SMOOTHER_L1DIAG;
643             else
644             { status = ERROR_INPUT_PAR; break; }
645             fscanf(fp, "%*[^\n]"); // skip rest of line
646         }
647
648         else if (strcmp(buffer,"AMG_smooth_order_v")==0) {
649             val = fscanf(fp,"%s",buffer);
650             if (val!=1 || strcmp(buffer,"=")!=0) {
651                 status = ERROR_INPUT_PAR; break;
652             }
653             val = fscanf(fp,"%s",buffer);
654             if (val!=1) { status = ERROR_INPUT_PAR; break; }
655
656             if ((strcmp(buffer,"NO")==0)||(strcmp(buffer,"no")==0))
657             Input->AMG_smooth_order_v = NO_ORDER;
658             else if ((strcmp(buffer,"CF")==0)||(strcmp(buffer,"cf")==0))
659             Input->AMG_smooth_order_v = CF_ORDER;
660             else
661             { status = ERROR_INPUT_PAR; break; }
662             fscanf(fp, "%*[^\n]"); // skip rest of line
663         }
664
665         else if (strcmp(buffer,"AMG_coarsening_type_v")==0) {
666             val = fscanf(fp,"%s",buffer);
667             if (val!=1 || strcmp(buffer,"=")!=0) {
668                 status = ERROR_INPUT_PAR; break;
669             }
670             val = fscanf(fp,"%d",&ibuff);
671             if (val!=1) { status = ERROR_INPUT_PAR; break; }
672             Input->AMG_coarsening_type_v = ibuff;
673             fscanf(fp, "%*[^\n]"); // skip rest of line
674         }
675
676         else if (strcmp(buffer,"AMG_interpolation_type_v")==0) {
677             val = fscanf(fp,"%s",buffer);
678             if (val!=1 || strcmp(buffer,"=")!=0) {
679                 status = ERROR_INPUT_PAR; break;
680             }
681             val = fscanf(fp,"%d",&ibuff);
682             if (val!=1) { status = ERROR_INPUT_PAR; break; }
683             Input->AMG_interpolation_type_v = ibuff;
684             fscanf(fp, "%*[^\n]"); // skip rest of line
685         }
686
687         else if (strcmp(buffer,"AMG_aggressive_level_v")==0) {
688             val = fscanf(fp,"%s",buffer);
689             if (val!=1 || strcmp(buffer,"=")!=0) {
690                 status = ERROR_INPUT_PAR; break;
691             }
692             val = fscanf(fp,"%d",&ibuff);
693             if (val!=1) { status = ERROR_INPUT_PAR; break; }
694             Input->AMG_aggressive_level_v = ibuff;
695             fscanf(fp, "%*[^\n]"); // skip rest of line
696         }
697
698         else if (strcmp(buffer,"AMG_aggressive_path_v")==0) {
699             val = fscanf(fp,"%s",buffer);
700             if (val!=1 || strcmp(buffer,"=")!=0) {
701                 status = ERROR_INPUT_PAR; break;
702             }
703             val = fscanf(fp,"%d",&ibuff);
704             if (val!=1) { status = ERROR_INPUT_PAR; break; }
705             Input->AMG_aggressive_path_v = ibuff;
706             fscanf(fp, "%*[^\n]"); // skip rest of line
707         }
708
709         else if (strcmp(buffer,"AMG_presmooth_iter_v")==0) {
710             val = fscanf(fp,"%s",buffer);
711             if (val!=1 || strcmp(buffer,"=")!=0) {
712                 status = ERROR_INPUT_PAR; break;
713             }
714             val = fscanf(fp,"%d",&ibuff);
```

```
715             if (val!=1) { status = ERROR_INPUT_PAR; break; }
716             Input->AMG_presmooth_iter_v = ibuff;
717             fscanf(fp, "%*[^\n]"); // skip rest of line
718         }
719
720         else if (strcmp(buffer,"AMG_postsmooth_iter_v")==0) {
721             val = fscanf(fp,"%s",buffer);
722             if (val!=1 || strcmp(buffer,"=")!=0) {
723                 status = ERROR_INPUT_PAR; break;
724             }
725             val = fscanf(fp,"%d",&ibuff);
726             if (val!=1) { status = ERROR_INPUT_PAR; break; }
727             Input->AMG_postsmooth_iter_v = ibuff;
728             fscanf(fp, "%*[^\n]"); // skip rest of line
729         }
730
731         else if (strcmp(buffer,"AMG_relaxation_v")==0) {
732             val = fscanf(fp,"%s",buffer);
733             if (val!=1 || strcmp(buffer,"=")!=0) {
734                 status = ERROR_INPUT_PAR; break;
735             }
736             val = fscanf(fp,"%lf",&dbuff);
737             if (val!=1) { status = ERROR_INPUT_PAR; break; }
738             Input->AMG_relaxation_v=dbuff;
739             fscanf(fp, "%*[^\n]"); // skip rest of line
740         }
741
742         else if (strcmp(buffer,"AMG_polynomial_degree_v")==0) {
743             val = fscanf(fp,"%s",buffer);
744             if (val!=1 || strcmp(buffer,"=")!=0) {
745                 status = ERROR_INPUT_PAR; break;
746             }
747             val = fscanf(fp,"%d",&ibuff);
748             if (val!=1) { status = ERROR_INPUT_PAR; break; }
749             Input->AMG_polynomial_degree_v = ibuff;
750             fscanf(fp, "%*[^\n]"); // skip rest of line
751         }
752
753         else if (strcmp(buffer,"AMG_strong_threshold_v")==0) {
754             val = fscanf(fp,"%s",buffer);
755             if (val!=1 || strcmp(buffer,"=")!=0) {
756                 status = ERROR_INPUT_PAR; break;
757             }
758             val = fscanf(fp,"%lf",&dbuff);
759             if (val!=1) { status = ERROR_INPUT_PAR; break; }
760             Input->AMG_strong_threshold_v = dbuff;
761             fscanf(fp, "%*[^\n]"); // skip rest of line
762         }
763
764         else if (strcmp(buffer,"AMG_truncation_threshold_v")==0) {
765             val = fscanf(fp,"%s",buffer);
766             if (val!=1 || strcmp(buffer,"=")!=0) {
767                 status = ERROR_INPUT_PAR; break;
768             }
769             val = fscanf(fp,"%lf",&dbuff);
770             if (val!=1) { status = ERROR_INPUT_PAR; break; }
771             Input->AMG_truncation_threshold_v = dbuff;
772             fscanf(fp, "%*[^\n]"); // skip rest of line
773         }
774
775         else if (strcmp(buffer,"AMG_max_row_sum_v")==0) {
776             val = fscanf(fp,"%s",buffer);
777             if (val!=1 || strcmp(buffer,"=")!=0) {
778                 status = ERROR_INPUT_PAR; break;
779             }
780             val = fscanf(fp,"%lf",&dbuff);
781             if (val!=1) { status = ERROR_INPUT_PAR; break; }
782             Input->AMG_max_row_sum_v = dbuff;
783             fscanf(fp, "%*[^\n]"); // skip rest of line
784         }
785
786         else if (strcmp(buffer,"AMG_amli_degree_v")==0) {
787             val = fscanf(fp,"%s",buffer);
788             if (val!=1 || strcmp(buffer,"=")!=0) {
789                 status = ERROR_INPUT_PAR; break;
790             }
791             val = fscanf(fp,"%d",&ibuff);
792             if (val!=1) { status = ERROR_INPUT_PAR; break; }
793             Input->AMG_amli_degree_v = ibuff;
794             fscanf(fp, "%*[^\n]"); // skip rest of line
795         }
796
797         else if (strcmp(buffer,"AMG_nl_amli_krylov_type_v")==0) {
798             val = fscanf(fp,"%s",buffer);
799             if (val!=1 || strcmp(buffer,"=")!=0) {
800                 status = ERROR_INPUT_PAR; break;
801             }
```

```
802             val = fscanf(fp,"%d",&ibuff);
803             if (val!=1) { status = ERROR_INPUT_PAR; break; }
804             Input->AMG_nl_amli_krylov_type_v = ibuff;
805             fscanf(fp, "%*[^\n]"); // skip rest of line
806         }
807
808         else if (strcmp(buffer,"AMG_ILU_levels_p")==0) {
809             val = fscanf(fp,"%s",buffer);
810             if (val!=1 || strcmp(buffer,"=")!=0) {
811                 status = ERROR_INPUT_PAR; break;
812             }
813             val = fscanf(fp,"%d",&ibuff);
814             if (val!=1) { status = ERROR_INPUT_PAR; break; }
815             Input->AMG_ILU_levels_p = ibuff;
816             fscanf(fp, "%*[^\n]"); // skip rest of line
817         }
818
819         else if (strcmp(buffer,"AMG_schwarz_levels_p")==0) {
820             val = fscanf(fp,"%s",buffer);
821             if (val!=1 || strcmp(buffer,"=")!=0) {
822                 status = ERROR_INPUT_PAR; break;
823             }
824             val = fscanf(fp,"%d",&ibuff);
825             if (val!=1) { status = FASP_SUCCESS; break; }
826             Input->AMG_schwarz_levels_p = ibuff;
827             fscanf(fp, "%*[^\n]"); // skip rest of line
828         }
829
830         else if (strcmp(buffer,"AMG_type_p")==0) {
831             val = fscanf(fp,"%s",buffer);
832             if (val!=1 || strcmp(buffer,"=")!=0) {
833                 status = ERROR_INPUT_PAR; break;
834             }
835             val = fscanf(fp,"%s",buffer);
836             if (val!=1) { status = ERROR_INPUT_PAR; break; }
837
838             if ((strcmp(buffer,"C")==0)||(strcmp(buffer,"c")==0))
839             Input->AMG_type_p = CLASSIC_AMG;
840             else if ((strcmp(buffer,"SA")==0)||(strcmp(buffer,"sa")==0))
841             Input->AMG_type_p = SA_AMG;
842             else if ((strcmp(buffer,"UA")==0)||(strcmp(buffer,"ua")==0))
843             Input->AMG_type_p = UA_AMG;
844             else
845             { status = ERROR_INPUT_PAR; break; }
846             fscanf(fp, "%*[^\n]"); // skip rest of line
847         }
848
849         else if (strcmp(buffer,"AMG_aggregation_type_p")==0) {
850             val = fscanf(fp,"%s",buffer);
851             if (val!=1 || strcmp(buffer,"=")!=0) {
852                 status = ERROR_INPUT_PAR; break;
853             }
854             val = fscanf(fp,"%d",&ibuff);
855             if (val!=1) { status = ERROR_INPUT_PAR; break; }
856             Input->AMG_aggregation_type_p = ibuff;
857             fscanf(fp, "%*[^\n]"); // skip rest of line
858         }
859
860         else if (strcmp(buffer,"AMG_pair_number_p")==0) {
861             val = fscanf(fp,"%s",buffer);
862             if (val!=1 || strcmp(buffer,"=")!=0) {
863                 status = ERROR_INPUT_PAR; break;
864             }
865             val = fscanf(fp,"%d",&ibuff);
866             if (val!=1) { status = ERROR_INPUT_PAR; break; }
867             Input->AMG_pair_number_p = ibuff;
868             fscanf(fp, "%*[^\n]"); // skip rest of line
869         }
870
871         else if (strcmp(buffer,"AMG_quality_bound_p")==0) {
872             val = fscanf(fp,"%s",buffer);
873             if (val!=1 || strcmp(buffer,"=")!=0) {
874                 status = ERROR_INPUT_PAR; break;
875             }
876             val = fscanf(fp,"%lf",&dbuff);
877             if (val!=1) { status = ERROR_INPUT_PAR; break; }
878             Input->AMG_quality_bound_p = dbuff;
879             fscanf(fp, "%*[^\n]"); // skip rest of line
880         }
881
882         else if (strcmp(buffer,"AMG_strong_coupled_p")==0) {
883             val = fscanf(fp,"%s",buffer);
884             if (val!=1 || strcmp(buffer,"=")!=0) {
885                 status = ERROR_INPUT_PAR; break;
886             }
887             val = fscanf(fp,"%lf",&dbuff);
888             if (val!=1) { status = ERROR_INPUT_PAR; break; }
```

```
889             Input->AMG_strong_coupled_p = dbuff;
890             fscanf(fp, "%*[^\n]"); // skip rest of line
891         }
892 
893         else if (strcmp(buffer,"AMG_max_aggregation_p")==0) {
894             val = fscanf(fp,"%s",buffer);
895             if (val!=1 || strcmp(buffer,"=")!=0) {
896                 status = ERROR_INPUT_PAR; break;
897             }
898             val = fscanf(fp,"%d",&ibuff);
899             if (val!=1) { status = ERROR_INPUT_PAR; break; }
900             Input->AMG_max_aggregation_p = ibuff;
901             fscanf(fp, "%*[^\n]"); // skip rest of line
902         }
903 
904         else if (strcmp(buffer,"AMG_tentative_smooth_p")==0) {
905             val = fscanf(fp,"%s",buffer);
906             if (val!=1 || strcmp(buffer,"=")!=0) {
907                 status = ERROR_INPUT_PAR; break;
908             }
909             val = fscanf(fp,"%lf",&dbuff);
910             if (val!=1) { status = ERROR_INPUT_PAR; break; }
911             Input->AMG_tentative_smooth_p = dbuff;
912             fscanf(fp, "%*[^\n]"); // skip rest of line
913         }
914 
915         else if (strcmp(buffer,"AMG_smooth_filter_p")==0) {
916             val = fscanf(fp,"%s",buffer);
917             if (val!=1 || strcmp(buffer,"=")!=0) {
918                 status = ERROR_INPUT_PAR; break;
919             }
920             val = fscanf(fp,"%s",buffer);
921             if (val!=1) { status = ERROR_INPUT_PAR; break; }
922 
923             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
924                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
925             Input->AMG_smooth_filter_p = ON;
926             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
927                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
928                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
929                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
930             Input->AMG_smooth_filter_p = OFF;
931             else
932             { status = ERROR_INPUT_PAR; break; }
933             fscanf(fp, "%*[^\n]"); // skip rest of line
934         }
935 
936         else if (strcmp(buffer,"AMG_coarse_scaling_p")==0) {
937             val = fscanf(fp,"%s",buffer);
938             if (val!=1 || strcmp(buffer,"=")!=0) {
939                 status = ERROR_INPUT_PAR; break;
940             }
941             val = fscanf(fp,"%s",buffer);
942             if (val!=1) { status = ERROR_INPUT_PAR; break; }
943 
944             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
945                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
946             Input->AMG_coarse_scaling_p = ON;
947             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
948                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
949                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
950                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
951             Input->AMG_coarse_scaling_p = OFF;
952             else
953             { status = ERROR_INPUT_PAR; break; }
954             fscanf(fp, "%*[^\n]"); // skip rest of line
955         }
956 
957         else if (strcmp(buffer,"AMG_levels_p")==0) {
958             val = fscanf(fp,"%s",buffer);
959             if (val!=1 || strcmp(buffer,"=")!=0) {
960                 status = ERROR_INPUT_PAR; break;
961             }
962             val = fscanf(fp,"%d",&ibuff);
963             if (val!=1) { status = ERROR_INPUT_PAR; break; }
964             Input->AMG_levels_p = ibuff;
965             fscanf(fp, "%*[^\n]"); // skip rest of line
966         }
967 
968         else if (strcmp(buffer,"AMG_tol_p")==0) {
969             val = fscanf(fp,"%s",buffer);
970             if (val!=1 || strcmp(buffer,"=")!=0) {
971                 status = ERROR_INPUT_PAR; break;
972             }
973             val = fscanf(fp,"%lf",&dbuff);
974             if (val!=1) { status = ERROR_INPUT_PAR; break; }
975             Input->AMG_tol_p = dbuff;
```

```
976             fscanf(fp, "%*[^\n]"); // skip rest of line
977         }
978
979         else if (strcmp(buffer,"AMG_maxit_p")==0) {
980             val = fscanf(fp,"%s",buffer);
981             if (val!=1 || strcmp(buffer,"=")!=0) {
982                 status = ERROR_INPUT_PAR; break;
983             }
984             val = fscanf(fp,"%d",&ibuff);
985             if (val!=1) { status = ERROR_INPUT_PAR; break; }
986             Input->AMG_maxit_p = ibuff;
987             fscanf(fp, "%*[^\n]"); // skip rest of line
988         }
989
990         else if (strcmp(buffer,"AMG_coarse_dof_p")==0) {
991             val = fscanf(fp,"%s",buffer);
992             if (val!=1 || strcmp(buffer,"=")!=0) {
993                 status = ERROR_INPUT_PAR; break;
994             }
995             val = fscanf(fp,"%d",&ibuff);
996             if (val!=1) { status = ERROR_INPUT_PAR; break; }
997             Input->AMG_coarse_dof_p = ibuff;
998             fscanf(fp, "%*[^\n]"); // skip rest of line
999         }
1000
1001         else if (strcmp(buffer,"AMG_coarse_solver_p")==0) {
1002             val = fscanf(fp,"%s",buffer);
1003             if (val!=1 || strcmp(buffer,"=")!=0) {
1004                 status = ERROR_INPUT_PAR; break;
1005             }
1006             val = fscanf(fp,"%d",&ibuff);
1007             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1008             Input->AMG_coarse_solver_p = ibuff;
1009             fscanf(fp, "%*[^\n]"); // skip rest of line
1010         }
1011
1012         else if (strcmp(buffer,"AMG_cycle_type_p")==0) {
1013             val = fscanf(fp,"%s",buffer);
1014             if (val!=1 || strcmp(buffer,"=")!=0) {
1015                 status = ERROR_INPUT_PAR; break;
1016             }
1017             val = fscanf(fp,"%s",buffer);
1018             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1019
1020             if ((strcmp(buffer,"V")==0)||(strcmp(buffer,"v")==0))
1021             Input->AMG_cycle_type_p = V_CYCLE;
1022             else if ((strcmp(buffer,"W")==0)||(strcmp(buffer,"w")==0))
1023             Input->AMG_cycle_type_p = W_CYCLE;
1024             else if ((strcmp(buffer,"A")==0)||(strcmp(buffer,"a")==0))
1025             Input->AMG_cycle_type_p = AMLI_CYCLE;
1026             else if ((strcmp(buffer,"NA")==0)||(strcmp(buffer,"na")==0))
1027             Input->AMG_cycle_type_p = NL_AMLI_CYCLE;
1028             else
1029             { status = ERROR_INPUT_PAR; break; }
1030             fscanf(fp, "%*[^\n]"); // skip rest of line
1031         }
1032
1033         else if (strcmp(buffer,"AMG_smoother_p")==0) {
1034             val = fscanf(fp,"%s",buffer);
1035             if (val!=1 || strcmp(buffer,"=")!=0) {
1036                 status = ERROR_INPUT_PAR; break;
1037             }
1038             val = fscanf(fp,"%s",buffer);
1039             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1040
1041             if ((strcmp(buffer,"JACOBI")==0)||(strcmp(buffer,"jacobi")==0))
1042             Input->AMG_smoother_p = SMOOTHER_JACOBI;
1043             else if ((strcmp(buffer,"GS")==0)||(strcmp(buffer,"gs")==0))
1044             Input->AMG_smoother_p = SMOOTHER_GS;
1045             else if ((strcmp(buffer,"SGS")==0)||(strcmp(buffer,"sgs")==0))
1046             Input->AMG_smoother_p = SMOOTHER_SGS;
1047             else if ((strcmp(buffer,"CG")==0)||(strcmp(buffer,"cg")==0))
1048             Input->AMG_smoother_p = SMOOTHER_CG;
1049             else if ((strcmp(buffer,"SOR")==0)||(strcmp(buffer,"sor")==0))
1050             Input->AMG_smoother_p = SMOOTHER_SOR;
1051             else if ((strcmp(buffer,"SSOR")==0)||(strcmp(buffer,"ssor")==0))
1052             Input->AMG_smoother_p = SMOOTHER_SSOR;
1053             else if ((strcmp(buffer,"GSOR")==0)||(strcmp(buffer,"gsor")==0))
1054             Input->AMG_smoother_p = SMOOTHER_GSOR;
1055             else if ((strcmp(buffer,"SGSOR")==0)||(strcmp(buffer,"sgsor")==0))
1056             Input->AMG_smoother_p = SMOOTHER_SGSOR;
1057             else if ((strcmp(buffer,"POLY")==0)||(strcmp(buffer,"poly")==0))
1058             Input->AMG_smoother_p = SMOOTHER_POLY;
1059             else if ((strcmp(buffer,"L1_DIAG")==0)||(strcmp(buffer,"l1_diag")==0))
1060             Input->AMG_smoother_p = SMOOTHER_L1DIAG;
1061             else
1062             { status = ERROR_INPUT_PAR; break; }
```

```
1063            fscanf(fp, "%*[^\n]"); // skip rest of line
1064        }
1065
1066        else if (strcmp(buffer,"AMG_smooth_order_p")==0) {
1067            val = fscanf(fp,"%s",buffer);
1068            if (val!=1 || strcmp(buffer,"=")!=0) {
1069                status = ERROR_INPUT_PAR; break;
1070            }
1071            val = fscanf(fp,"%s",buffer);
1072            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1073
1074            if ((strcmp(buffer,"NO")==0)||(strcmp(buffer,"no")==0))
1075            Input->AMG_smooth_order_p = NO_ORDER;
1076            else if ((strcmp(buffer,"CF")==0)||(strcmp(buffer,"cf")==0))
1077            Input->AMG_smooth_order_p = CF_ORDER;
1078            else
1079            { status = ERROR_INPUT_PAR; break; }
1080            fscanf(fp, "%*[^\n]"); // skip rest of line
1081        }
1082
1083        else if (strcmp(buffer,"AMG_coarsening_type_p")==0) {
1084            val = fscanf(fp,"%s",buffer);
1085            if (val!=1 || strcmp(buffer,"=")!=0) {
1086                status = ERROR_INPUT_PAR; break;
1087            }
1088            val = fscanf(fp,"%d",&ibuff);
1089            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1090            Input->AMG_coarsening_type_p = ibuff;
1091            fscanf(fp, "%*[^\n]"); // skip rest of line
1092        }
1093
1094        else if (strcmp(buffer,"AMG_interpolation_type_p")==0) {
1095            val = fscanf(fp,"%s",buffer);
1096            if (val!=1 || strcmp(buffer,"=")!=0) {
1097                status = ERROR_INPUT_PAR; break;
1098            }
1099            val = fscanf(fp,"%d",&ibuff);
1100            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1101            Input->AMG_interpolation_type_p = ibuff;
1102            fscanf(fp, "%*[^\n]"); // skip rest of line
1103        }
1104
1105        else if (strcmp(buffer,"AMG_aggressive_level_p")==0) {
1106            val = fscanf(fp,"%s",buffer);
1107            if (val!=1 || strcmp(buffer,"=")!=0) {
1108                status = ERROR_INPUT_PAR; break;
1109            }
1110            val = fscanf(fp,"%d",&ibuff);
1111            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1112            Input->AMG_aggressive_level_p = ibuff;
1113            fscanf(fp, "%*[^\n]"); // skip rest of line
1114        }
1115
1116        else if (strcmp(buffer,"AMG_aggressive_path_p")==0) {
1117            val = fscanf(fp,"%s",buffer);
1118            if (val!=1 || strcmp(buffer,"=")!=0) {
1119                status = ERROR_INPUT_PAR; break;
1120            }
1121            val = fscanf(fp,"%d",&ibuff);
1122            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1123            Input->AMG_aggressive_path_p = ibuff;
1124            fscanf(fp, "%*[^\n]"); // skip rest of line
1125        }
1126
1127        else if (strcmp(buffer,"AMG_presmooth_iter_p")==0) {
1128            val = fscanf(fp,"%s",buffer);
1129            if (val!=1 || strcmp(buffer,"=")!=0) {
1130                status = ERROR_INPUT_PAR; break;
1131            }
1132            val = fscanf(fp,"%d",&ibuff);
1133            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1134            Input->AMG_presmooth_iter_p = ibuff;
1135            fscanf(fp, "%*[^\n]"); // skip rest of line
1136        }
1137
1138        else if (strcmp(buffer,"AMG_postsmooth_iter_p")==0) {
1139            val = fscanf(fp,"%s",buffer);
1140            if (val!=1 || strcmp(buffer,"=")!=0) {
1141                status = ERROR_INPUT_PAR; break;
1142            }
1143            val = fscanf(fp,"%d",&ibuff);
1144            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1145            Input->AMG_postsmooth_iter_p = ibuff;
1146            fscanf(fp, "%*[^\n]"); // skip rest of line
1147        }
1148
1149        else if (strcmp(buffer,"AMG_relaxation_p")==0) {
```

```
1150            val = fscanf(fp,"%s",buffer);
1151            if (val!=1 || strcmp(buffer,"=")!=0) {
1152                status = ERROR_INPUT_PAR; break;
1153            }
1154            val = fscanf(fp,"%lf",&dbuff);
1155            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1156            Input->AMG_relaxation_p=dbuff;
1157            fscanf(fp, "%*[^\n]"); // skip rest of line
1158        }
1159
1160        else if (strcmp(buffer,"AMG_polynomial_degree_p")==0) {
1161            val = fscanf(fp,"%s",buffer);
1162            if (val!=1 || strcmp(buffer,"=")!=0) {
1163                status = ERROR_INPUT_PAR; break;
1164            }
1165            val = fscanf(fp,"%d",&ibuff);
1166            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1167            Input->AMG_polynomial_degree_p = ibuff;
1168            fscanf(fp, "%*[^\n]"); // skip rest of line
1169        }
1170
1171        else if (strcmp(buffer,"AMG_strong_threshold_p")==0) {
1172            val = fscanf(fp,"%s",buffer);
1173            if (val!=1 || strcmp(buffer,"=")!=0) {
1174                status = ERROR_INPUT_PAR; break;
1175            }
1176            val = fscanf(fp,"%lf",&dbuff);
1177            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1178            Input->AMG_strong_threshold_p = dbuff;
1179            fscanf(fp, "%*[^\n]"); // skip rest of line
1180        }
1181
1182        else if (strcmp(buffer,"AMG_truncation_threshold_p")==0) {
1183            val = fscanf(fp,"%s",buffer);
1184            if (val!=1 || strcmp(buffer,"=")!=0) {
1185                status = ERROR_INPUT_PAR; break;
1186            }
1187            val = fscanf(fp,"%lf",&dbuff);
1188            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1189            Input->AMG_truncation_threshold_p = dbuff;
1190            fscanf(fp, "%*[^\n]"); // skip rest of line
1191        }
1192
1193        else if (strcmp(buffer,"AMG_max_row_sum_p")==0) {
1194            val = fscanf(fp,"%s",buffer);
1195            if (val!=1 || strcmp(buffer,"=")!=0) {
1196                status = ERROR_INPUT_PAR; break;
1197            }
1198            val = fscanf(fp,"%lf",&dbuff);
1199            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1200            Input->AMG_max_row_sum_p = dbuff;
1201            fscanf(fp, "%*[^\n]"); // skip rest of line
1202        }
1203
1204        else if (strcmp(buffer,"AMG_amli_degree_p")==0) {
1205            val = fscanf(fp,"%s",buffer);
1206            if (val!=1 || strcmp(buffer,"=")!=0) {
1207                status = ERROR_INPUT_PAR; break;
1208            }
1209            val = fscanf(fp,"%d",&ibuff);
1210            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1211            Input->AMG_amli_degree_p = ibuff;
1212            fscanf(fp, "%*[^\n]"); // skip rest of line
1213        }
1214
1215        else if (strcmp(buffer,"AMG_nl_amli_krylov_type_p")==0) {
1216            val = fscanf(fp,"%s",buffer);
1217            if (val!=1 || strcmp(buffer,"=")!=0) {
1218                status = ERROR_INPUT_PAR; break;
1219            }
1220            val = fscanf(fp,"%d",&ibuff);
1221            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1222            Input->AMG_nl_amli_krylov_type_p = ibuff;
1223            fscanf(fp, "%*[^\n]"); // skip rest of line
1224        }
1225
1226        else if (strcmp(buffer,"ILU_type")==0) {
1227            val = fscanf(fp,"%s",buffer);
1228            if (val!=1 || strcmp(buffer,"=")!=0) {
1229                status = ERROR_INPUT_PAR; break;
1230            }
1231            val = fscanf(fp,"%d",&ibuff);
1232            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1233            Input->ILU_type = ibuff;
1234            fscanf(fp, "%*[^\n]"); // skip rest of line
1235        }
1236
```

```
1237         else if (strcmp(buffer,"ILU_lfil")==0) {
1238             val = fscanf(fp,"%s",buffer);
1239             if (val!=1 || strcmp(buffer,"=")!=0) {
1240                 status = ERROR_INPUT_PAR; break;
1241             }
1242             val = fscanf(fp,"%d",&ibuff);
1243             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1244             Input->ILU_lfil = ibuff;
1245             fscanf(fp, "%*[^\n]"); // skip rest of line
1246         }
1247 
1248         else if (strcmp(buffer,"ILU_droptol")==0) {
1249             val = fscanf(fp,"%s",buffer);
1250             if (val!=1 || strcmp(buffer,"=")!=0) {
1251                 status = ERROR_INPUT_PAR; break;
1252             }
1253             val = fscanf(fp,"%lf",&dbuff);
1254             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1255             Input->ILU_droptol = dbuff;
1256             fscanf(fp, "%*[^\n]"); // skip rest of line
1257         }
1258 
1259         else if (strcmp(buffer,"ILU_relax")==0) {
1260             val = fscanf(fp,"%s",buffer);
1261             if (val!=1 || strcmp(buffer,"=")!=0) {
1262                 status = ERROR_INPUT_PAR; break;
1263             }
1264             val = fscanf(fp,"%lf",&dbuff);
1265             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1266             Input->ILU_relax = dbuff;
1267             fscanf(fp, "%*[^\n]"); // skip rest of line
1268         }
1269 
1270         else if (strcmp(buffer,"ILU_permtol")==0) {
1271             val = fscanf(fp,"%s",buffer);
1272             if (val!=1 || strcmp(buffer,"=")!=0) {
1273                 status = ERROR_INPUT_PAR; break;
1274             }
1275             val = fscanf(fp,"%lf",&dbuff);
1276             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1277             Input->ILU_permtol = dbuff;
1278             fscanf(fp, "%*[^\n]"); // skip rest of line
1279         }
1280 
1281         else if (strcmp(buffer,"SWZ_mmsize")==0) {
1282             val = fscanf(fp,"%s",buffer);
1283             if (val!=1 || strcmp(buffer,"=")!=0) {
1284                 status = ERROR_INPUT_PAR; break;
1285             }
1286             val = fscanf(fp,"%d",&ibuff);
1287             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1288             Input->SWZ_mmsize = ibuff;
1289             fscanf(fp, "%*[^\n]"); // skip rest of line
1290         }
1291 
1292         else if (strcmp(buffer,"SWZ_maxlvl")==0) {
1293             val = fscanf(fp,"%s",buffer);
1294             if (val!=1 || strcmp(buffer,"=")!=0) {
1295                 status = ERROR_INPUT_PAR; break;
1296             }
1297             val = fscanf(fp,"%d",&ibuff);
1298             if (val!=1) {status = ERROR_INPUT_PAR; break; }
1299             Input->SWZ_maxlvl = ibuff;
1300             fscanf(fp, "%*[^\n]"); // skip rest of line
1301         }
1302 
1303         else if (strcmp(buffer,"SWZ_type")==0) {
1304             val = fscanf(fp,"%s",buffer);
1305             if (val!=1 || strcmp(buffer,"=")!=0) {
1306                 status = ERROR_INPUT_PAR; break;
1307             }
1308             val = fscanf(fp,"%d",&ibuff);
1309             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1310             Input->SWZ_type = ibuff;
1311             fscanf(fp, "%*[^\n]"); // skip rest of line
1312         }
1313 
1314         else {
1315             printf("### WARNING: Unknown input keyword %s!\n", buffer);
1316             fscanf(fp, "%*[^\n]"); // skip rest of line
1317         }
1318     }
1319 
1320     fclose(fp);
1321 
1322     // if meet unexpected input, stop the program
1323     fasp_chkerr(status, __FUNCTION__);
```

```
1324
1325     // sanity checks
1326     status = fasp_ns_param_check(Input);
1327
1328 #if DEBUG_MODE > 1
1329     printf("### DEBUG: Reading input status = %d\n", status);
1330 #endif
1331
1332     fasp_chkerr(status, __FUNCTION__);
1333 }
```

4.3 AuxParam.c File Reference

Initialize, set, or print input data and parameters.

```
#include <stdio.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- void fasp_ns_param_amg_set (AMG_ns_param *param, input_ns_param *inparam)
  Set AMG_param from INPUT.
- void fasp_ns_param_solver_init (itsolver_ns_param *itsparam)
  Initialize AMG parameters.
- void fasp_ns_param_ilu_set (ILU_param *iluparam, input_ns_param *inparam)
  Set ILU_param with INPUT.
- void fasp_ns_param_swz_set (SWZ_param *swzparam, input_ns_param *inparam)
  Set SWZ_param with INPUT.

4.3.1 Detailed Description

Initialize, set, or print input data and parameters.

Note

This file contains Level-0 (Aux) functions.

4.3.2 Function Documentation

4.3.2.1 fasp_ns_param_amg_set()

```
void fasp_ns_param_amg_set (
            AMG_ns_param * param,
            input_ns_param * inparam )
```

Set AMG_param from INPUT.

Parameters

param	Parameters for AMG
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Modified by Xiaozhe Hu on 02/21/2014

Definition at line 230 of file AuxParam.c.

```
232 {
233     // iterative solver parameter for the velocity block
234     param->param_v.AMG_type    = inparam->AMG_type_v;
235     param->param_v.print_level = inparam->print_level;
236
237     if (inparam->itsolver_type_v == SOLVER_AMG) {
238         param->param_v.maxit = inparam->pre_maxit_v;
239         param->param_v.tol   = inparam->pre_tol_v;
240     }
241     else if (inparam->itsolver_type_v == SOLVER_FMG) {
242         param->param_v.maxit = inparam->pre_maxit_v;
243         param->param_v.tol   = inparam->pre_tol_v;
244     }
245     else {
246         param->param_v.maxit = inparam->AMG_maxit_v;
247         param->param_v.tol   = inparam->AMG_tol_v;
248     }
249
250     param->param_v.max_levels           = inparam->AMG_levels_v;
251     param->param_v.cycle_type           = inparam->AMG_cycle_type_v;
252     param->param_v.smoother             = inparam->AMG_smoother_v;
253     param->param_v.smooth_order         = inparam->AMG_smooth_order_v;
254     param->param_v.relaxation           = inparam->AMG_relaxation_v;
255     param->param_v.polynomial_degree    = inparam->AMG_polynomial_degree_v;
256     param->param_v.presmooth_iter       = inparam->AMG_presmooth_iter_v;
257     param->param_v.postsmooth_iter      = inparam->AMG_postsmooth_iter_v;
258     param->param_v.coarse_dof           = inparam->AMG_coarse_dof_v;
259     param->param_v.coarse_solver        = inparam->AMG_coarse_solver_v;
260     param->param_v.coarse_scaling       = inparam->AMG_coarse_scaling_v;
261     param->param_v.amli_degree          = inparam->AMG_amli_degree_v;
262     param->param_v.amli_coef            = NULL;
263     param->param_v.nl_amli_krylov_type  = inparam->AMG_nl_amli_krylov_type_v;
264
265     param->param_v.coarsening_type      = inparam->AMG_coarsening_type_v;
266     param->param_v.interpolation_type   = inparam->AMG_interpolation_type_v;
267     param->param_v.strong_threshold     = inparam->AMG_strong_threshold_v;
268     param->param_v.truncation_threshold = inparam->AMG_truncation_threshold_v;
269     param->param_v.max_row_sum          = inparam->AMG_max_row_sum_v;
270     param->param_v.aggressive_level     = inparam->AMG_aggressive_level_v;
271     param->param_v.aggressive_path      = inparam->AMG_aggressive_path_v;
272
273     param->param_v.aggregation_type     = inparam->AMG_aggregation_type_v;
274     param->param_v.pair_number          = inparam->AMG_pair_number_v;
275     param->param_v.quality_bound        = inparam->AMG_quality_bound_v;
276
277     param->param_v.strong_coupled       = inparam->AMG_strong_coupled_v;
278     param->param_v.max_aggregation      = inparam->AMG_max_aggregation_v;
279     param->param_v.tentative_smooth     = inparam->AMG_tentative_smooth_v;
280     param->param_v.smooth_filter        = inparam->AMG_smooth_filter_v;
281
282     param->param_v.ILU_levels           = inparam->AMG_ILU_levels_v;
283     param->param_v.ILU_type             = inparam->ILU_type;
284     param->param_v.ILU_lfil             = inparam->ILU_lfil;
285     param->param_v.ILU_droptol          = inparam->ILU_droptol;
286     param->param_v.ILU_relax            = inparam->ILU_relax;
287     param->param_v.ILU_permtol          = inparam->ILU_permtol;
288     param->param_v.SWZ_levels           = inparam->AMG_schwarz_levels_v;
```

```
289     param->param_v.SWZ_mmsize           = inparam->SWZ_mmsize;
290     param->param_v.SWZ_maxlvl           = inparam->SWZ_maxlvl;
291     param->param_v.SWZ_type             = inparam->SWZ_type;
292
293     // iterative solver parameter for the pressure block
294     param->param_p.AMG_type    = inparam->AMG_type_p;
295     param->param_p.print_level = inparam->print_level;
296
297     if (inparam->itsolver_type_p == SOLVER_AMG) {
298         param->param_p.maxit   = inparam->pre_maxit_p;
299         param->param_p.tol     = inparam->pre_tol_p;
300     }
301     else if (inparam->itsolver_type_p == SOLVER_FMG) {
302         param->param_p.maxit   = inparam->pre_maxit_p;
303         param->param_p.tol     = inparam->pre_tol_p;
304     }
305     else {
306         param->param_p.maxit   = inparam->AMG_maxit_p;
307         param->param_p.tol     = inparam->AMG_tol_p;
308     }
309
310     param->param_p.max_levels           = inparam->AMG_levels_p;
311     param->param_p.cycle_type           = inparam->AMG_cycle_type_p;
312     param->param_p.smoother             = inparam->AMG_smoother_p;
313     param->param_p.smooth_order         = inparam->AMG_smooth_order_p;
314     param->param_p.relaxation           = inparam->AMG_relaxation_p;
315     param->param_p.polynomial_degree    = inparam->AMG_polynomial_degree_p;
316     param->param_p.presmooth_iter       = inparam->AMG_presmooth_iter_p;
317     param->param_p.postsmooth_iter      = inparam->AMG_postsmooth_iter_p;
318     param->param_p.coarse_dof           = inparam->AMG_coarse_dof_p;
319     param->param_p.coarse_solver        = inparam->AMG_coarse_solver_p;
320     param->param_p.coarse_scaling       = inparam->AMG_coarse_scaling_p;
321     param->param_p.amli_degree          = inparam->AMG_amli_degree_p;
322     param->param_p.amli_coef            = NULL;
323     param->param_p.nl_amli_krylov_type  = inparam->AMG_nl_amli_krylov_type_p;
324
325     param->param_p.coarsening_type      = inparam->AMG_coarsening_type_p;
326     param->param_p.interpolation_type   = inparam->AMG_interpolation_type_p;
327     param->param_p.strong_threshold     = inparam->AMG_strong_threshold_p;
328     param->param_p.truncation_threshold = inparam->AMG_truncation_threshold_p;
329     param->param_p.max_row_sum          = inparam->AMG_max_row_sum_p;
330     param->param_p.aggressive_level     = inparam->AMG_aggressive_level_p;
331     param->param_p.aggressive_path      = inparam->AMG_aggressive_path_p;
332
333     param->param_p.aggregation_type     = inparam->AMG_aggregation_type_p;
334     param->param_p.pair_number          = inparam->AMG_pair_number_p;
335     param->param_p.quality_bound        = inparam->AMG_quality_bound_p;
336
337     param->param_p.strong_coupled       = inparam->AMG_strong_coupled_p;
338     param->param_p.max_aggregation      = inparam->AMG_max_aggregation_p;
339     param->param_p.tentative_smooth     = inparam->AMG_tentative_smooth_p;
340     param->param_p.smooth_filter        = inparam->AMG_smooth_filter_p;
341
342     param->param_p.ILU_levels           = inparam->AMG_ILU_levels_p;
343     param->param_p.ILU_type             = inparam->ILU_type;
344     param->param_p.ILU_lfil             = inparam->ILU_lfil;
345     param->param_p.ILU_droptol          = inparam->ILU_droptol;
346     param->param_p.ILU_relax            = inparam->ILU_relax;
347     param->param_p.ILU_permtol          = inparam->ILU_permtol;
348     param->param_p.SWZ_levels           = inparam->AMG_schwarz_levels_p;
349     param->param_p.SWZ_mmsize           = inparam->SWZ_mmsize;
350     param->param_p.SWZ_maxlvl           = inparam->SWZ_maxlvl;
351     param->param_p.SWZ_type             = inparam->SWZ_type;
352 }
```

4.3.2.2 fasp_ns_param_ilu_set()

```
void fasp_ns_param_ilu_set (
             ILU_param * iluparam,
             input_ns_param * inparam )
```

Set ILU_param with INPUT.

Parameters

iluparam	Parameters for ILU
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Definition at line 473 of file AuxParam.c.

```
475 {
476     iluparam->print_level = inparam->print_level;
477     iluparam->ILU_type    = inparam->ILU_type;
478     iluparam->ILU_lfil    = inparam->ILU_lfil;
479     iluparam->ILU_droptol = inparam->ILU_droptol;
480     iluparam->ILU_relax   = inparam->ILU_relax;
481     iluparam->ILU_permtol = inparam->ILU_permtol;
482 }
```

4.3.2.3 fasp_ns_param_solver_init()

```
void fasp_ns_param_solver_init (
             itsolver_ns_param * itsparam )
```

Initialize AMG parameters.

Parameters

amgparam	Parameters for AMG

Author

Lu Wang

Date

2014/02/11

Modified by Xiaozhe Hu on 02/21/2014

Definition at line 366 of file AuxParam.c.

```
367 {
368     itsparam->itsolver_type   = SOLVER_CG;
369     itsparam->precond_type    = PREC_AMG;
370     itsparam->stop_type       = STOP_REL_RES;
371     itsparam->maxit           = 100;
372     itsparam->tol             = 1e-8;
373     itsparam->restart         = 20;
374     itsparam->print_level     = 0;
375
376     // iterative solver parameter for the velocity block
377     itsparam->itsolver_type_v = SOLVER_CG;
378     itsparam->precond_type_v  = PREC_AMG;
379     itsparam->pre_maxit_v     = 20;
380     itsparam->pre_tol_v       = 1e-2;
381     itsparam->pre_restart_v   = 20;
382     itsparam->print_level_v   = 0;
383
384     // iterative solver parameter for the pressure block
385     itsparam->itsolver_type_p = SOLVER_CG;
386     itsparam->precond_type_p  = PREC_AMG;
387     itsparam->pre_maxit_p     = 20;
388     itsparam->pre_tol_p       = 1e-2;
389     itsparam->pre_restart_p   = 20;
390     itsparam->print_level_p   = 0;
391 }
```

4.3.2.4 fasp_ns_param_swz_set()

```
void fasp_ns_param_swz_set (
             SWZ_param * swzparam,
             input_ns_param * inparam )
```

Set SWZ_param with INPUT.

Parameters

swzparam	Parameters for Schwarz method
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Definition at line 495 of file AuxParam.c.

```
497 {
498     swzparam->print_level = inparam->print_level;
499     swzparam->SWZ_type    = inparam->SWZ_type;
500     swzparam->SWZ_maxlvl  = inparam->SWZ_maxlvl;
501     swzparam->SWZ_mmsize  = inparam->SWZ_mmsize;
502 }
```

4.4 basisP0.inl File Reference

Basis functions and problem information.

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

Functions

- void basis (double nodes[3][2], double s, int index, double phi[2])

  basis function of Lagrange element, i.e. area coordiante
- double area (double x1, double x2, double x3, double y1, double y2, double y3)

  get area for triangle p1(x1,y1),p2(x2,y2),p3(x3,y3)
- void localb (double(∗nodes)[2], double ∗b)

  get local right-hand side b from triangle nodes

4.4.1 Detailed Description

Basis functions and problem information.

4.4.2 Function Documentation

4.4.2.1 area()

```
double area (
             double x1,
             double x2,
             double x3,
             double y1,
             double y2,
             double y3 )
```

get area for triangle p1(x1,y1),p2(x2,y2),p3(x3,y3)

Parameters

x1	the x-axis value of the point p1
x2	the x-axis value of the point p2
x3	the x-axis value of the point p3
y1	the y-axis value of the point p1
y2	the y-axis value of the point p2
y3	the y-axis value of the point p3

Returns

area of the trianle

Author

Xuehai Huang

Date

03/29/2009

Parameters

x1	the x-axis value of the point p1
x2	the x-axis value of the point p2
x3	the x-axis value of the point p3
y1	the y-axis value of the point p1
y2	the y-axis value of the point p2
y3	the y-axis value of the point p3

Returns

area of the trianle

Author

Lu Wang

Date

11/12/2011

Definition at line 56 of file basisP0.inl.

```
57 {
58     return ((x2-x1)*(y3-y1)-(y2-y1)*(x3-x1))/2;
59 }
```

4.4.2.2 basis()

```
void basis (
              double nodes[3][2],
              double s,
              int index,
              double phi[2] )
```

basis function of Lagrange element, i.e. area coordiante

basis function of area coordiante

Parameters

nodes[3][2]	the vertice of the triangule
s	the area of the triangule
index	the indicator of the basis function
phi[2]	basis function

Returns

void

Author

Xuehai Huang

Date

03/29/2009

Parameters

nodes[3][2]	the vertice of the triangule
s	the area of the triangule
index	the indicator of the basis function
phi[2]	basis function

Returns

void

Author

Lu Wang

Date

11/13/2011

Definition at line 34 of file basisP0.inl.

```
35 {
36     const int node1 = (index+1)%3, node2 = (index+2)%3;
37     phi[0]=(nodes[node1][1]-nodes[node2][1])/(2.0*s);
38     phi[1]=(nodes[node2][0]-nodes[node1][0])/(2.0*s);
39 }
```

4.4.2.3 localb()

```
void localb (
             double(*) nodes[2],
             double * b )
```

get local right-hand side b from triangle nodes

Parameters

(∗nodes)[2]	the vertice of the triangule
∗b	local right-hand side

Author

Xuehai Huang

Date

03/29/2009

Definition at line 71 of file basisP0.inl.

```
72 {
73     const double s=2.0*area(nodes[0][0],nodes[1][0],nodes[2][0],nodes[0][1],nodes[1][1],nodes[2][1]);
74     const int num_qp=16; // the number of numerical intergation points
75     double x,y,a;
76     double gauss[num_qp][3];
77     int i;
78 
79     fasp_init_Gauss(num_qp, 2, gauss); // gauss intergation initial
80 
81     for (i=0;i<3;++i) b[i]=0;
82 
83     for (i=0;i<num_qp;++i) {
84         x=nodes[0][0]*gauss[i][0]+nodes[1][0]*gauss[i][1]+nodes[2][0]*(1-gauss[i][0]-gauss[i][1]);
85         y=nodes[0][1]*gauss[i][0]+nodes[1][1]*gauss[i][1]+nodes[2][1]*(1-gauss[i][0]-gauss[i][1]);
86         a=f(x,y);
87 
88         b[0]+=a*gauss[i][2]*gauss[i][0];
89         b[1]+=a*gauss[i][2]*gauss[i][1];
90         b[2]+=a*gauss[i][2]*(1-gauss[i][0]-gauss[i][1]);
91     }
92 
93     b[0]*=s; b[1]*=s; b[2]*=s;
94 }
```

4.5 basisP2.inl File Reference

Basis functions and problem information.

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

4.5.1 Detailed Description

Basis functions and problem information.

4.6 BlaIO.c File Reference

I/O functions for NS solvers.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- void fasp_dblc_read (char ∗fileA, char ∗fileB, char ∗fileC, char ∗filerhs, dBLCmat ∗A, dvector ∗r)

  Read E and rhs from file in block_dSTRmat format.
- void fasp_dblc_read_ruth (char ∗fileA, char ∗fileB, char ∗fileC, char ∗fileD, char ∗filerhs, char ∗filex0, dBL↩ Cmat ∗A, dvector ∗r, dvector ∗x0)

  Read E and rhs from file in block_dSTRmat format.

4.6.1 Detailed Description

I/O functions for NS solvers.

Note

This file contains Level-1 (Bla) functions.

// TODO: Fix Doxygen. –Chensong // TODO: Remove unused functions. –Chensong

4.6.2 Function Documentation

4.6.2.1 fasp_dblc_read()

```
void fasp_dblc_read (
             char * fileA,
             char * fileB,
             char * fileC,
             char * filerhs,
             dBLCmat * A,
             dvector * r )
```

Read E and rhs from file in block_dSTRmat format.

Parameters

fileA	file name of A
fileB	file name of B
fileC	file name of C
fileArhs	file name of right hand side
A	pointer to the dBLCmat

Note

E = (A B^T) (B C) File format: This routine reads a dCSRmat matrix from files in the following format:

Author

Lu WANG

Date

02/24/2012

Definition at line 45 of file BlaIO.c.

```
51 {
52     int numA,nnz,numB,nnzb,nnzc;
53     int i, k, n;
54     int ivalue,wall;
55     double value;
56 
57     // read file A
58     FILE *fp=fopen(fileA,"r");
59     if ( fp == NULL ) {
60         printf("### ERROR: Opening file %s failed!\n", fileA);
61         exit(ERROR_OPEN_FILE);
62     }
63     printf("%s: reading file %s...\n", __FUNCTION__, fileA);
64 
65     wall = fscanf(fp,"%d %d",&numA,&nnz); // read dimension of the problem
66     fasp_dcsr_alloc (numA,numA,nnz,A->blocks[0]);
67     // read matrix A
68     for (i=0;i<numA+1;++i) {
69         wall = fscanf(fp, "%d", &ivalue);
70         A->blocks[0]->IA[i]=ivalue;
71     }
72     for (i=0;i<nnz;++i) {
73         wall = fscanf(fp, "%d", &ivalue);
74         A->blocks[0]->JA[i]=ivalue-1;
75     }
76     for (i=0;i<nnz;++i) {
77         wall = fscanf(fp, "%le", &value);
78         A->blocks[0]->val[i]=value;
79     }
80     fclose(fp);
81 
82     fp=fopen(fileB,"r");
83     if ( fp == NULL ) {
84         printf("### ERROR: Opening file %s failed!\n",fileB);
85         exit(ERROR_OPEN_FILE);
86     }
87     printf("%s: reading file %s...\n", __FUNCTION__, fileB);
88 
89     wall = fscanf(fp,"%d %d",&numB,&nnzb); // read dimension of the problem
90     fasp_dcsr_alloc (numB,numA,nnzb,A->blocks[2]);
91 
92     // read matrix B
93     for (i=0;i<numB+1;++i) {
94         wall = fscanf(fp, "%d", &ivalue);
95         A->blocks[2]->IA[i]=ivalue;
96     }
97 
98     for (i=0;i<nnzb;++i) {
```

```
99          wall = fscanf(fp, "%d", &ivalue);
100          A->blocks[2]->JA[i]=ivalue-1;
101      }
102
103      for (i=0;i<nnzb;++i) {
104          wall = fscanf(fp, "%le", &value);
105          A->blocks[2]->val[i]=value;
106      }
107      fclose(fp);
108      fasp_dcsr_trans(A->blocks[2],A->blocks[1]);
109
110      fp=fopen(fileC,"r");
111      if ( fp == NULL ) {
112          printf("### ERROR: Opening file %s failed!\n",fileC);
113          exit(ERROR_OPEN_FILE);
114      }
115      printf("%s: reading file %s...\n", __FUNCTION__, fileC);
116
117      wall = fscanf(fp,"%d %d",&numB,&nnzc); // read dimension of the problem
118      fasp_dcsr_alloc (numB,numB,nnzc,A->blocks[3]);
119
120      // read matrix B
121      for (i=0;i<numB+1;++i) {
122          wall = fscanf(fp, "%d", &ivalue);
123          A->blocks[3]->IA[i]=ivalue;
124      }
125
126      for (i=0;i<nnzc;++i) {
127          wall = fscanf(fp, "%d", &ivalue);
128          A->blocks[3]->JA[i]=ivalue-1;
129      }
130
131      for (i=0;i<nnzc;++i) {
132          wall = fscanf(fp, "%le", &value);
133          A->blocks[3]->val[i]=value;
134      }
135      fclose(fp);
136
137      fp=fopen(filerhs,"r");
138      if ( fp == NULL ) {
139          printf("### ERROR: Opening file %s failed!\n",filerhs);
140          exit(ERROR_OPEN_FILE);
141      }
142      printf("%s: reading file %s...\n", __FUNCTION__, filerhs);
143
144      fasp_dvec_alloc (numA+numB,r);
145      for (i=0;i<numA+numB;++i) {
146          wall = fscanf(fp, "%le", &value);
147          r->val[i]=value;
148      }
149      fclose(fp);
150  }
```

4.6.2.2 fasp_dblc_read_ruth()

```
void fasp_dblc_read_ruth (
              char * fileA,
              char * fileB,
              char * fileC,
              char * fileD,
              char * filerhs,
              char * filex0,
              dBLCmat * A,
              dvector * r,
              dvector * x0 )
```

Read E and rhs from file in block_dSTRmat format.

Parameters

fileA	file name of A

Parameters

fileB	file name of B
fileC	file name of C
fileArhs	file name of right hand side
A	pointer to the dBLCmat

Note

E = (A B^T) (B C) File format: This routine reads a dCSRmat matrix from files in the following format:

Author

Lu WANG

Date

02/24/2012

Definition at line 301 of file BlaIO.c.

```
310 {
311     fasp_dcoo_read (fileA,A->blocks[0]);
312     fasp_dcoo_read (fileB,A->blocks[1]);
313     fasp_dcoo_read (fileC,A->blocks[2]);
314     fasp_dcoo_read (fileD,A->blocks[3]);
315     fasp_dvec_read (filerhs,r);
316     fasp_dvec_read (filex0,x0);
317 }
```

4.7 fasp4ns.h File Reference

Main header file for FASP4NS package.

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

Data Structures

- struct AMG_ns_data

  Data for AMG solvers for Navier-Stokes problems.
- struct itsolver_ns_param

  Parameters passed to iterative solvers.
- struct precond_ns_param

  Parameters passed to the preconditioner for generalized Navier-Stokes problems.
- struct precond_ns_data

  Data passed to the preconditioner for generalized Navier-Stokes problems.
- struct precond_pnp_stokes_data

  Data passed to the preconditioner for block preconditioning for dBLCmat format.

Typedefs

- typedef struct precond_ns_param precond_ns_param
  Parameters passed to the preconditioner for generalized Navier-Stokes problems.
- typedef struct precond_ns_data precond_ns_data
  Data passed to the preconditioner for generalized Navier-Stokes problems.

4.7.1 Detailed Description

Main header file for FASP4NS package.

Note

: modified by Xiaozhe Hu on Feb. 21, 2014
: modified by Xiaozhe Hu on May. 27, 2014

4.8 fasp4ns_functs.h File Reference

Function decoration for the FASP package.

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

Functions

- SHORT fasp_ns_param_check (const input_ns_param *inparam)
  Simple check on input parameters.
- void fasp_ns_param_input (char *filenm, input_ns_param *Input)
  Read input parameters for NS problem from disk file.
- void fasp_ns_param_amg_set (AMG_ns_param *param, input_ns_param *inparam)
  Set AMG_param from INPUT.
- void fasp_ns_param_solver_init (itsolver_ns_param *itsparam)
  Initialize AMG parameters.
- void fasp_ns_param_ilu_set (ILU_param *iluparam, input_ns_param *inparam)
  Set ILU_param with INPUT.
- void fasp_ns_param_swz_set (SWZ_param *swzparam, input_ns_param *inparam)
  Set SWZ_param with INPUT.
- void fasp_dblc_read (char *fileA, char *fileB, char *fileC, char *filerhs, dBLCmat *A, dvector *r)
  Read E and rhs from file in block_dSTRmat format.
- void fasp_dblc_read_ruth (char *fileA, char *fileB, char *fileC, char *fileD, char *filerhs, char *filex0, dBL↩
  Cmat *A, dvector *r, dvector *x0)
  Read E and rhs from file in block_dSTRmat format.
- void fasp_precond_ns_bdiag (REAL *r, REAL *z, void *data)
  block diagonal preconditioning for ns equation
- void fasp_precond_ns_low_btri (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_up_btri (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_blu (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_simple (REAL *r, REAL *z, void *data)

- void fasp_precond_ns_simpler (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_uzawa (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_projection (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_DGS (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_LSCDGS (REAL *r, REAL *z, void *data)
- void fasp_precond_pnp_stokes_diag (REAL *r, REAL *z, void *data)

  block diagonal preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_lower (REAL *r, REAL *z, void *data)

  block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_upper (REAL *r, REAL *z, void *data)

  block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_diag_inexact (REAL *r, REAL *z, void *data)

  block diagonal preconditioning (3x3 block matrix, each diagonal block is solved inexactly)
- void fasp_precond_pnp_stokes_lower_inexact (REAL *r, REAL *z, void *data)

  block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_upper_inexact (REAL *r, REAL *z, void *data)

  block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- SHORT fasp_solver_dblc_krylov_navier_stokes (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam)

  Solve Ax=b by standard Krylov methods for NS equations.
- SHORT fasp_solver_dblc_krylov_navier_stokes_pmass (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam, dCSRmat *Mp)

  Solve Ax=b by standard Krylov methods for NS equations.
- SHORT fasp_solver_dblc_krylov_navier_stokes_schur_pmass (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam, dCSRmat *Mp)

  Solve Ax=b by standard Krylov methods for NS equations.
- INT fasp_solver_dblc_krylov_pnp_stokes (dBLCmat *A, dvector *b, dvector *x, ITS_param *itparam, ITS_param *itparam_pnp, AMG_param *amgparam_pnp, itsolver_ns_param *itparam_stokes, AMG_ns_param *amgparam_stokes, const int num_velocity, const int num_pressure)

  Solve Ax = b by standard Krylov methods.
- void fasp_fwrapper_krylov_navier_stokes_nsym_ (INT *nA, INT *nnzA, INT *ia, INT *ja, REAL *aval, INT *nB, INT *mB, INT *nnzB, INT *ib, INT *jb, REAL *bval, INT *nC, INT *mC, INT *nnzC, INT *ic, INT *jc, REAL *cval, REAL *b, REAL *u)

  Solve [A B; C O] u = b by Krylov method with block preconditioners.
- void fasp_fwrapper_krylov_navier_stokes_sym_ (INT *nA, INT *nnzA, INT *ia, INT *ja, REAL *aval, INT *nB, INT *nnzB, INT *ib, INT *jb, REAL *bval, INT *nC, INT *nnzC, INT *ic, INT *jc, REAL *cval, REAL *b, REAL *u)

  Solve [A B'; B C] u = b by Krylov method with block preconditioners.

4.8.1 Detailed Description

Function decoration for the FASP package.

Warning

DO NOT EDIT!!! This file is automatically generated!

4.8.2 Function Documentation

4.8.2.1 fasp_dblc_read()

```
void fasp_dblc_read (
             char * fileA,
             char * fileB,
             char * fileC,
             char * filerhs,
             dBLCmat * A,
             dvector * r )
```

Read E and rhs from file in block_dSTRmat format.

Parameters

fileA	file name of A
fileB	file name of B
fileC	file name of C
fileArhs	file name of right hand side
A	pointer to the dBLCmat

Note

E = (A B^T) (B C) File format: This routine reads a dCSRmat matrix from files in the following format:

Author

Lu WANG

Date

02/24/2012

Definition at line 45 of file BlaIO.c.

```
51 {
52     int numA,nnz,numB,nnzb,nnzc;
53     int i, k, n;
54     int ivalue,wall;
55     double value;
56
57     // read file A
58     FILE *fp=fopen(fileA,"r");
59     if ( fp == NULL ) {
60         printf("### ERROR: Opening file %s failed!\n", fileA);
61         exit(ERROR_OPEN_FILE);
62     }
63     printf("%s: reading file %s...\n", __FUNCTION__, fileA);
64
65     wall = fscanf(fp,"%d %d",&numA,&nnz); // read dimension of the problem
66     fasp_dcsr_alloc (numA,numA,nnz,A->blocks[0]);
67     // read matrix A
68     for (i=0;i<numA+1;++i) {
69         wall = fscanf(fp, "%d", &ivalue);
70         A->blocks[0]->IA[i]=ivalue;
```

```
71      }
72      for (i=0;i<nnz;++i) {
73          wall = fscanf(fp, "%d", &ivalue);
74          A->blocks[0]->JA[i]=ivalue-1;
75      }
76      for (i=0;i<nnz;++i) {
77          wall = fscanf(fp, "%le", &value);
78          A->blocks[0]->val[i]=value;
79      }
80      fclose(fp);
81
82      fp=fopen(fileB,"r");
83      if ( fp == NULL ) {
84          printf("### ERROR: Opening file %s failed!\n",fileB);
85          exit(ERROR_OPEN_FILE);
86      }
87      printf("%s: reading file %s...\n", __FUNCTION__, fileB);
88
89      wall = fscanf(fp,"%d %d",&numB,&nnzb); // read dimension of the problem
90      fasp_dcsr_alloc (numB,numA,nnzb,A->blocks[2]);
91
92      // read matrix B
93      for (i=0;i<numB+1;++i) {
94          wall = fscanf(fp, "%d", &ivalue);
95          A->blocks[2]->IA[i]=ivalue;
96      }
97
98      for (i=0;i<nnzb;++i) {
99          wall = fscanf(fp, "%d", &ivalue);
100          A->blocks[2]->JA[i]=ivalue-1;
101      }
102
103      for (i=0;i<nnzb;++i) {
104          wall = fscanf(fp, "%le", &value);
105          A->blocks[2]->val[i]=value;
106      }
107      fclose(fp);
108      fasp_dcsr_trans(A->blocks[2],A->blocks[1]);
109
110      fp=fopen(fileC,"r");
111      if ( fp == NULL ) {
112          printf("### ERROR: Opening file %s failed!\n",fileC);
113          exit(ERROR_OPEN_FILE);
114      }
115      printf("%s: reading file %s...\n", __FUNCTION__, fileC);
116
117      wall = fscanf(fp,"%d %d",&numB,&nnzc); // read dimension of the problem
118      fasp_dcsr_alloc (numB,numB,nnzc,A->blocks[3]);
119
120      // read matrix B
121      for (i=0;i<numB+1;++i) {
122          wall = fscanf(fp, "%d", &ivalue);
123          A->blocks[3]->IA[i]=ivalue;
124      }
125
126      for (i=0;i<nnzc;++i) {
127          wall = fscanf(fp, "%d", &ivalue);
128          A->blocks[3]->JA[i]=ivalue-1;
129      }
130
131      for (i=0;i<nnzc;++i) {
132          wall = fscanf(fp, "%le", &value);
133          A->blocks[3]->val[i]=value;
134      }
135      fclose(fp);
136
137      fp=fopen(filerhs,"r");
138      if ( fp == NULL ) {
139          printf("### ERROR: Opening file %s failed!\n",filerhs);
140          exit(ERROR_OPEN_FILE);
141      }
142      printf("%s: reading file %s...\n", __FUNCTION__, filerhs);
143
144      fasp_dvec_alloc (numA+numB,r);
145      for (i=0;i<numA+numB;++i) {
146          wall = fscanf(fp, "%le", &value);
147          r->val[i]=value;
148      }
149      fclose(fp);
150  }
```

4.8.2.2 fasp_dblc_read_ruth()

```
void fasp_dblc_read_ruth (
             char * fileA,
             char * fileB,
             char * fileC,
             char * fileD,
             char * filerhs,
             char * filex0,
             dBLCmat * A,
             dvector * r,
             dvector * x0 )
```

Read E and rhs from file in block_dSTRmat format.

Parameters

fileA	file name of A
fileB	file name of B
fileC	file name of C
fileArhs	file name of right hand side
A	pointer to the dBLCmat

Note

E = (A B^T) (B C) File format: This routine reads a dCSRmat matrix from files in the following format:

Author

Lu WANG

Date

02/24/2012

Definition at line 301 of file BlaIO.c.

```
310 {
311     fasp_dcoo_read (fileA,A->blocks[0]);
312     fasp_dcoo_read (fileB,A->blocks[1]);
313     fasp_dcoo_read (fileC,A->blocks[2]);
314     fasp_dcoo_read (fileD,A->blocks[3]);
315     fasp_dvec_read (filerhs,r);
316     fasp_dvec_read (filex0,x0);
317 }
```

4.8.2.3 fasp_fwrapper_krylov_navier_stokes_nsym_()

```
void fasp_fwrapper_krylov_navier_stokes_nsym_ (
             INT * nA,
             INT * nnzA,
             INT * ia,
             INT * ja,
             REAL * aval,
             INT * nB,
             INT * mB,
             INT * nnzB,
             INT * ib,
             INT * jb,
             REAL * bval,
             INT * nC,
             INT * mC,
             INT * nnzC,
             INT * ic,
             INT * jc,
             REAL * cval,
             REAL * b,
             REAL * u )
```

Solve [A B; C O] u = b by Krylov method with block preconditioners.

Parameters

nA	num of rows/cols of A
nnzA	num of nonzeros of A
ia	IA of A in CSR format
ja	JA of A in CSR format
aval	VAL of A in CSR format
nB	num of rows of B
mB	num of cols of B
nnzB	num of nonzeros of B
ib	IA of B in CSR format
jb	JA of B in CSR format
bval	VAL of B in CSR format
nC	num of rows of C
mC	num of cols of C
nnzC	num of nonzeros of C
ic	IA of C in CSR format
jc	JA of C in CSR format
cval	VAL of C in CSR format
b	rhs vector
u	solution vector

Author

Lu Wang

Date

03/20/2014

Modified by Chensong Zhang on 03/16/2018 Step 0. Read input parameters

Definition at line 61 of file SolWrapper.c.

```
80 {
81     dBLCmat A; // coefficient matrix
82     dCSRmat matA11, matA21, matA12, matA22;
83     dvector rhs, sol; // right-hand-side, solution
84     precond_ns_param psparam; // parameters for ns precond
85     precond_ns_data  psdata; // data for ns precond
86     int i,flag;
87 
88     char *inputfile = "ini/ns.dat";
89     input_ns_param     inparam;  // parameters from input files
90     itsolver_ns_param  itparam;  // parameters for itsolver
91     AMG_ns_param       amgparam; // parameters for AMG
92     ILU_param          iluparam; // parameters for ILU
93     SWZ_param          swzparam; // parameters for Schwarz
94 
96     fasp_ns_param_input(inputfile, &inparam);
97     fasp_ns_param_init(&inparam, &itparam, &amgparam, &iluparam, &swzparam);
98 
99     // Set local parameters
100     const int print_level   = inparam.print_level;
101     const int problem_num   = inparam.problem_num;
102     const int itsolver_type = inparam.solver_type;
103     const int precond_type  = inparam.precond_type;
104 
105 #if DEBUG_MODE > 0
106     printf("### DEBUG: nA = %d\n", *nA);
107     printf("### DEBUG: nB = %d, mB = %d\n", *nB, *mB);
108     printf("### DEBUG: nC = %d, mc = %d\n", *nC, *mC);
109 #endif
110 
111     // initialize dBLCmat pointer
112     A.brow = 2; A.bcol = 2;
113     A.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
114     if ( A.blocks == NULL ) {
115         printf("### ERROR: Cannot allocate memory %s!\n", __FUNCTION__);
116         exit(ERROR_ALLOC_MEM);
117     }
118     A.blocks[0] = &matA11;
119     A.blocks[1] = &matA12;
120     A.blocks[2] = &matA21;
121     A.blocks[3] = &matA22;
122 
123     // initialize matrix
124     matA11.row = *nA; matA11.col = *nA; matA11.nnz = *nnzA;
125     matA11.IA  = ia;  matA11.JA  = ja;  matA11.val = aval;
126 
127     matA12.row = *nB; matA12.col = *mB; matA12.nnz = *nnzB;
128     matA12.IA  = ib;  matA12.JA  = jb;  matA12.val = bval;
129 
130     matA21.row = *nC; matA21.col = *mC; matA21.nnz = *nnzC;
131     matA21.IA  = ic;  matA21.JA  = jc;  matA21.val = cval;
132 
133     // generate an empty matrix
134     fasp_dcsr_alloc(*nC,*nC,1,&matA22);
135 
136     // shift the index to start from 0 (for C routines)
137     for ( i=0; i<matA11.row+1; i++ ) matA11.IA[i]--;
138     for ( i=0; i<matA12.row+1; i++ ) matA12.IA[i]--;
139     for ( i=0; i<matA21.row+1; i++ ) matA21.IA[i]--;
140     for ( i=0; i<matA11.nnz;   i++ ) matA11.JA[i]--;
141     for ( i=0; i<matA12.nnz;   i++ ) matA12.JA[i]--;
142     for ( i=0; i<matA21.nnz;   i++ ) matA21.JA[i]--;
143 
144     // initialize rhs and sol vectors
145     rhs.row = *nA+*nC; rhs.val = b;
146     sol.row = *nA+*nC; sol.val = u;
147 
148     if (print_level>0) {
149         printf("Max it num = %d\n", inparam.itsolver_maxit);
150         printf("Tolerance  = %e\n", inparam.itsolver_tol);
151     }
152 
153     flag = fasp_solver_dblc_krylov_navier_stokes(&A, &rhs, &sol, &
    itparam,
154                                                  &amgparam, &iluparam, &swzparam);
155 }
```

4.8.2.4 fasp_fwrapper_krylov_navier_stokes_sym_()

```
void fasp_fwrapper_krylov_navier_stokes_sym_ (
             INT * nA,
             INT * nnzA,
             INT * ia,
             INT * ja,
             REAL * aval,
             INT * nB,
             INT * nnzB,
             INT * ib,
             INT * jb,
             REAL * bval,
             INT * nC,
             INT * nnzC,
             INT * ic,
             INT * jc,
             REAL * cval,
             REAL * b,
             REAL * u )
```

Solve [A B'; B C] u = b by Krylov method with block preconditioners.

Parameters

nA	num of cols of A
nnzA	num of nonzeros of A
ia	IA of A in CSR format
ja	JA of A in CSR format
aval	VAL of A in CSR format
nB	num of cols of B
nnzB	num of nonzeros of B
ib	IA of B in CSR format
jb	JA of B in CSR format
bval	VAL of B in CSR format
nC	num of cols of C
nnzC	num of nonzeros of C
ic	IA of C in CSR format
jc	JA of C in CSR format
cval	VAL of C in CSR format
b	rhs vector
u	solution vector

Author

Lu Wang

Date

03/14/2012

Modified by Chensong Zhang on 03/13/2018 Step 0. Read input parameters

Definition at line 189 of file SolWrapper.c.

```
206 {
207     dBLCmat A; // coefficient matrix
208     dCSRmat matA11, matA21, matA12, matA22;
209     dvector rhs, sol; // right-hand-side, solution
210     precond_ns_param psparam; // parameters for ns precond
211     precond_ns_data  psdata; // data for ns precond
212     int i, flag;
213 
215     char *inputfile = "ini/ns.dat";
216     input_ns_param     inparam;  // parameters from input files
217     itsolver_ns_param  itparam;  // parameters for itsolver
218     AMG_ns_param       amgparam; // parameters for AMG
219     ILU_param          iluparam; // parameters for ILU
220     SWZ_param          swzparam; // parameters for Schwarz
221 
222     fasp_ns_param_input(inputfile, &inparam);
223     fasp_ns_param_init(&inparam, &itparam, &amgparam, &iluparam, &swzparam);
224 
225     // set local parameters
226     const int print_level   = inparam.print_level;
227     const int problem_num   = inparam.problem_num;
228     const int itsolver_type = inparam.solver_type;
229     const int precond_type  = inparam.precond_type;
230 
231 #if DEBUG_MODE > 0
232     printf("### DEBUG: nA = %d, nB = %d, nC = %d\n", *nA, *nB, *nC);
233 #endif
234 
235     // initialize dBLCmat pointer
236     A.brow = 2; A.bcol = 2;
237     A.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
238     if ( A.blocks == NULL ) {
239         printf("### ERROR: Cannot allocate memory %s!\n", __FUNCTION__);
240         exit(ERROR_ALLOC_MEM);
241     }
242     A.blocks[0] = &matA11;
243     A.blocks[1] = &matA12;
244     A.blocks[2] = &matA21;
245     A.blocks[3] = &matA22;
246 
247     // initialize matrix
248     matA11.row = *nA; matA11.col = *nA; matA11.nnz = *nnzA;
249     matA11.IA  = ia;  matA11.JA  = ja;  matA11.val = aval;
250 
251     matA21.row = *nB; matA21.col = *nA; matA21.nnz = *nnzB;
252     matA21.IA  = ib;  matA21.JA  = jb;  matA21.val = bval;
253 
254     matA22.row = *nC; matA22.col = *nC; matA22.nnz = *nnzC;
255     matA22.IA  = ic;  matA22.JA  = jc;  matA22.val = cval;
256 
257     // shift the index to start from 0 (for C routines)
258     for ( i=0; i<matA11.row+1; i++ ) matA11.IA[i]--;
259     for ( i=0; i<matA21.row+1; i++ ) matA21.IA[i]--;
260     for ( i=0; i<matA22.row+1; i++ ) matA22.IA[i]--;
261     for ( i=0; i<matA11.nnz;   i++ ) matA11.JA[i]--;
262     for ( i=0; i<matA21.nnz;   i++ ) matA21.JA[i]--;
263     for ( i=0; i<matA22.nnz;   i++ ) matA22.JA[i]--;
264 
265     // get transform of B
266     fasp_dcsr_trans(&matA21, &matA12);
267 
268     rhs.row = *nA + *nB; rhs.val = b;
269     sol.row = *nA + *nB; sol.val = u;
270 
271     if (print_level>0) {
272         printf("Max it num = %d\n", inparam.itsolver_maxit);
273         printf("Tolerance  = %e\n", inparam.itsolver_tol);
274     }
275 
276     flag = fasp_solver_dblc_krylov_navier_stokes(&A, &rhs, &sol, &
    itparam,
277                                                  &amgparam, &iluparam, &swzparam);
278 }
```

4.8.2.5 fasp_ns_param_amg_set()

```
void fasp_ns_param_amg_set (
             AMG_ns_param * param,
             input_ns_param * inparam )
```

Set AMG_param from INPUT.

Parameters

param	Parameters for AMG
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Modified by Xiaozhe Hu on 02/21/2014

Definition at line 230 of file AuxParam.c.

```
232 {
233     // iterative solver parameter for the velocity block
234     param->param_v.AMG_type    = inparam->AMG_type_v;
235     param->param_v.print_level = inparam->print_level;
236
237     if (inparam->itsolver_type_v == SOLVER_AMG) {
238         param->param_v.maxit = inparam->pre_maxit_v;
239         param->param_v.tol   = inparam->pre_tol_v;
240     }
241     else if (inparam->itsolver_type_v == SOLVER_FMG) {
242         param->param_v.maxit = inparam->pre_maxit_v;
243         param->param_v.tol   = inparam->pre_tol_v;
244     }
245     else {
246         param->param_v.maxit = inparam->AMG_maxit_v;
247         param->param_v.tol   = inparam->AMG_tol_v;
248     }
249
250     param->param_v.max_levels           = inparam->AMG_levels_v;
251     param->param_v.cycle_type           = inparam->AMG_cycle_type_v;
252     param->param_v.smoother             = inparam->AMG_smoother_v;
253     param->param_v.smooth_order         = inparam->AMG_smooth_order_v;
254     param->param_v.relaxation           = inparam->AMG_relaxation_v;
255     param->param_v.polynomial_degree    = inparam->AMG_polynomial_degree_v;
256     param->param_v.presmooth_iter       = inparam->AMG_presmooth_iter_v;
257     param->param_v.postsmooth_iter      = inparam->AMG_postsmooth_iter_v;
258     param->param_v.coarse_dof           = inparam->AMG_coarse_dof_v;
259     param->param_v.coarse_solver        = inparam->AMG_coarse_solver_v;
260     param->param_v.coarse_scaling       = inparam->AMG_coarse_scaling_v;
261     param->param_v.amli_degree          = inparam->AMG_amli_degree_v;
262     param->param_v.amli_coef            = NULL;
263     param->param_v.nl_amli_krylov_type  = inparam->AMG_nl_amli_krylov_type_v;
264
265     param->param_v.coarsening_type      = inparam->AMG_coarsening_type_v;
266     param->param_v.interpolation_type   = inparam->AMG_interpolation_type_v;
267     param->param_v.strong_threshold     = inparam->AMG_strong_threshold_v;
268     param->param_v.truncation_threshold = inparam->AMG_truncation_threshold_v;
269     param->param_v.max_row_sum          = inparam->AMG_max_row_sum_v;
270     param->param_v.aggressive_level     = inparam->AMG_aggressive_level_v;
271     param->param_v.aggressive_path      = inparam->AMG_aggressive_path_v;
272
273     param->param_v.aggregation_type     = inparam->AMG_aggregation_type_v;
274     param->param_v.pair_number          = inparam->AMG_pair_number_v;
275     param->param_v.quality_bound        = inparam->AMG_quality_bound_v;
276
277     param->param_v.strong_coupled       = inparam->AMG_strong_coupled_v;
278     param->param_v.max_aggregation      = inparam->AMG_max_aggregation_v;
279     param->param_v.tentative_smooth     = inparam->AMG_tentative_smooth_v;
280     param->param_v.smooth_filter        = inparam->AMG_smooth_filter_v;
281
282     param->param_v.ILU_levels           = inparam->AMG_ILU_levels_v;
283     param->param_v.ILU_type             = inparam->ILU_type;
284     param->param_v.ILU_lfil             = inparam->ILU_lfil;
285     param->param_v.ILU_droptol          = inparam->ILU_droptol;
286     param->param_v.ILU_relax            = inparam->ILU_relax;
287     param->param_v.ILU_permtol          = inparam->ILU_permtol;
288     param->param_v.SWZ_levels           = inparam->AMG_schwarz_levels_v;
```

```
289     param->param_v.SWZ_mmsize           = inparam->SWZ_mmsize;
290     param->param_v.SWZ_maxlvl           = inparam->SWZ_maxlvl;
291     param->param_v.SWZ_type             = inparam->SWZ_type;
292
293     // iterative solver parameter for the pressure block
294     param->param_p.AMG_type    = inparam->AMG_type_p;
295     param->param_p.print_level = inparam->print_level;
296
297     if (inparam->itsolver_type_p == SOLVER_AMG) {
298         param->param_p.maxit   = inparam->pre_maxit_p;
299         param->param_p.tol     = inparam->pre_tol_p;
300     }
301     else if (inparam->itsolver_type_p == SOLVER_FMG) {
302         param->param_p.maxit   = inparam->pre_maxit_p;
303         param->param_p.tol     = inparam->pre_tol_p;
304     }
305     else {
306         param->param_p.maxit   = inparam->AMG_maxit_p;
307         param->param_p.tol     = inparam->AMG_tol_p;
308     }
309
310     param->param_p.max_levels           = inparam->AMG_levels_p;
311     param->param_p.cycle_type           = inparam->AMG_cycle_type_p;
312     param->param_p.smoother             = inparam->AMG_smoother_p;
313     param->param_p.smooth_order         = inparam->AMG_smooth_order_p;
314     param->param_p.relaxation           = inparam->AMG_relaxation_p;
315     param->param_p.polynomial_degree    = inparam->AMG_polynomial_degree_p;
316     param->param_p.presmooth_iter       = inparam->AMG_presmooth_iter_p;
317     param->param_p.postsmooth_iter      = inparam->AMG_postsmooth_iter_p;
318     param->param_p.coarse_dof           = inparam->AMG_coarse_dof_p;
319     param->param_p.coarse_solver        = inparam->AMG_coarse_solver_p;
320     param->param_p.coarse_scaling       = inparam->AMG_coarse_scaling_p;
321     param->param_p.amli_degree          = inparam->AMG_amli_degree_p;
322     param->param_p.amli_coef            = NULL;
323     param->param_p.nl_amli_krylov_type  = inparam->AMG_nl_amli_krylov_type_p;
324
325     param->param_p.coarsening_type      = inparam->AMG_coarsening_type_p;
326     param->param_p.interpolation_type   = inparam->AMG_interpolation_type_p;
327     param->param_p.strong_threshold     = inparam->AMG_strong_threshold_p;
328     param->param_p.truncation_threshold = inparam->AMG_truncation_threshold_p;
329     param->param_p.max_row_sum          = inparam->AMG_max_row_sum_p;
330     param->param_p.aggressive_level     = inparam->AMG_aggressive_level_p;
331     param->param_p.aggressive_path      = inparam->AMG_aggressive_path_p;
332
333     param->param_p.aggregation_type     = inparam->AMG_aggregation_type_p;
334     param->param_p.pair_number          = inparam->AMG_pair_number_p;
335     param->param_p.quality_bound        = inparam->AMG_quality_bound_p;
336
337     param->param_p.strong_coupled       = inparam->AMG_strong_coupled_p;
338     param->param_p.max_aggregation      = inparam->AMG_max_aggregation_p;
339     param->param_p.tentative_smooth     = inparam->AMG_tentative_smooth_p;
340     param->param_p.smooth_filter        = inparam->AMG_smooth_filter_p;
341
342     param->param_p.ILU_levels           = inparam->AMG_ILU_levels_p;
343     param->param_p.ILU_type             = inparam->ILU_type;
344     param->param_p.ILU_lfil             = inparam->ILU_lfil;
345     param->param_p.ILU_droptol          = inparam->ILU_droptol;
346     param->param_p.ILU_relax            = inparam->ILU_relax;
347     param->param_p.ILU_permtol          = inparam->ILU_permtol;
348     param->param_p.SWZ_levels           = inparam->AMG_schwarz_levels_p;
349     param->param_p.SWZ_mmsize           = inparam->SWZ_mmsize;
350     param->param_p.SWZ_maxlvl           = inparam->SWZ_maxlvl;
351     param->param_p.SWZ_type             = inparam->SWZ_type;
352 }
```

4.8.2.6 fasp_ns_param_check()

```
SHORT fasp_ns_param_check (
             const input_ns_param * inparam )
```

Simple check on input parameters.

Parameters

inparam	Input parameters

Author

Chensong Zhang

Date

09/29/2013

Modified by Xiaozhe Hu on 05/27/2014 Modified by Chensong Zhang on 03/18/2018

Definition at line 36 of file AuxInput.c.

```
37 {
38     SHORT status = FASP_SUCCESS;
39
40     if ( inparam->problem_num<0
41          || inparam->solver_type<0
42          || inparam->solver_type>50
43          || inparam->precond_type<0
44          || inparam->itsolver_tol<=0
45          || inparam->itsolver_maxit<=0
46          || inparam->stop_type<=0
47          || inparam->stop_type>3
48          || inparam->restart<0
49          || inparam->ILU_type<=0
50          || inparam->ILU_type>3
51          || inparam->ILU_lfil<0
52          || inparam->ILU_droptol<=0
53          || inparam->ILU_relax<0
54          || inparam->ILU_permtol<0
55          || inparam->SWZ_mmsize<0
56          || inparam->SWZ_maxlvl<0
57          || inparam->SWZ_type<0
58          || inparam->AMG_type_v<=0
59          || inparam->AMG_type_v>3
60          || inparam->AMG_cycle_type_v<=0
61          || inparam->AMG_cycle_type_v>4
62          || inparam->AMG_levels_v<0
63          || inparam->AMG_ILU_levels_v<0
64          || inparam->AMG_coarse_dof_v<=0
65          || inparam->AMG_tol_v<0
66          || inparam->AMG_maxit_v<0
67          || inparam->AMG_coarsening_type_v<=0
68          || inparam->AMG_coarsening_type_v>4
69          || inparam->AMG_interpolation_type_v<0
70          || inparam->AMG_interpolation_type_v>5
71          || inparam->AMG_smoother_v<0
72          || inparam->AMG_smoother_v>20
73          || inparam->AMG_strong_threshold_v<0.0
74          || inparam->AMG_strong_threshold_v>0.9999
75          || inparam->AMG_truncation_threshold_v<0.0
76          || inparam->AMG_truncation_threshold_v>0.9999
77          || inparam->AMG_max_row_sum_v<0.0
78          || inparam->AMG_presmooth_iter_v<0
79          || inparam->AMG_postsmooth_iter_v<0
80          || inparam->AMG_amli_degree_v<0
81          || inparam->AMG_aggressive_level_v<0
82          || inparam->AMG_aggressive_path_v<0
83          || inparam->AMG_strong_coupled_v<0
84          || inparam->AMG_max_aggregation_v<=0
85          || inparam->AMG_tentative_smooth_v<0
86          || inparam->AMG_smooth_filter_v<0
87          || inparam->AMG_type_p<=0
88          || inparam->AMG_type_p>3
89          || inparam->AMG_cycle_type_p<=0
90          || inparam->AMG_cycle_type_p>4
91          || inparam->AMG_levels_p<0
92          || inparam->AMG_ILU_levels_p<0
93          || inparam->AMG_coarse_dof_p<=0
94          || inparam->AMG_tol_p<0
95          || inparam->AMG_maxit_p<0
96          || inparam->AMG_coarsening_type_p<=0
97          || inparam->AMG_coarsening_type_p>4
98          || inparam->AMG_interpolation_type_p<0
99          || inparam->AMG_interpolation_type_p>5
100         || inparam->AMG_smoother_p<0
101         || inparam->AMG_smoother_p>20
102         || inparam->AMG_strong_threshold_p<0.0
103         || inparam->AMG_strong_threshold_p>0.9999
104         || inparam->AMG_truncation_threshold_p<0.0
```

```
105         || inparam->AMG_truncation_threshold_p>0.9999
106         || inparam->AMG_max_row_sum_p<0.0
107         || inparam->AMG_presmooth_iter_p<0
108         || inparam->AMG_postsmooth_iter_p<0
109         || inparam->AMG_amli_degree_p<0
110         || inparam->AMG_aggressive_level_p<0
111         || inparam->AMG_aggressive_path_p<0
112         || inparam->AMG_strong_coupled_p<0
113         || inparam->AMG_max_aggregation_p<=0
114         || inparam->AMG_tentative_smooth_p<0
115         || inparam->AMG_smooth_filter_p<0
116         ) status = ERROR_INPUT_PAR;
117
118     return status;
119 }
```

4.8.2.7 fasp_ns_param_ilu_set()

```
void fasp_ns_param_ilu_set (
             ILU_param * iluparam,
             input_ns_param * inparam )
```

Set ILU_param with INPUT.

Parameters

iluparam	Parameters for ILU
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Definition at line 473 of file AuxParam.c.

```
475 {
476     iluparam->print_level = inparam->print_level;
477     iluparam->ILU_type    = inparam->ILU_type;
478     iluparam->ILU_lfil    = inparam->ILU_lfil;
479     iluparam->ILU_droptol = inparam->ILU_droptol;
480     iluparam->ILU_relax   = inparam->ILU_relax;
481     iluparam->ILU_permtol = inparam->ILU_permtol;
482 }
```

4.8.2.8 fasp_ns_param_input()

```
void fasp_ns_param_input (
             char * filenm,
             input_ns_param * Input )
```

Read input parameters for NS problem from disk file.

Parameters

filenm	File name for input file
Input	Input parameters

Author

Lu Wang

Date

02/15/2012

Modified by Chensong Zhang on 03/27/2017: check unexpected error Modified by Chensong Zhang on 09/23/2017: new skip the line Modified by Chensong Zhang on 03/18/2018: format

Definition at line 136 of file AuxInput.c.

```
138 {
139     char     buffer[500]; // Note: max number of char for each line!
140     INT      val;
141     SHORT    status = FASP_SUCCESS;
142
143     // set default input parameters
144     fasp_ns_param_input_init(Input);
145
146     // if input file is not specified, use the default values
147     if (filenm==NULL) return;
148
149     FILE *fp = fopen(filenm,"r");
150     if (fp==NULL) {
151         printf("### ERROR: Could not open file %s...\n", filenm);
152         exit(ERROR_OPEN_FILE);
153     }
154
155     while ( status == FASP_SUCCESS ) {
156         INT   ibuff;
157         REAL  dbuff;
158         char  sbuff[500];
159
160         val = fscanf(fp,"%s",buffer);
161
162         if (val==EOF) break;
163         if (val!=1) { status = ERROR_INPUT_PAR; break; }
164         if (buffer[0]=='[' || buffer[0]=='%' || buffer[0]=='|') {
165             fscanf(fp, "%*[^\n]"); // skip rest of line
166             continue;
167         }
168
169         // match keyword and scan for value
170         if (strcmp(buffer,"workdir")==0) {
171             val = fscanf(fp,"%s",buffer);
172             if (val!=1 || strcmp(buffer,"=")!=0) {
173                 status = ERROR_INPUT_PAR; break;
174             }
175             val = fscanf(fp,"%s",sbuff);
176             if (val!=1) { status = ERROR_INPUT_PAR; break; }
177             strncpy(Input->workdir,sbuff,128);
178             fscanf(fp, "%*[^\n]"); // skip rest of line
179         }
180
181         else if (strcmp(buffer,"problem_num")==0) {
182             val = fscanf(fp,"%s",buffer);
183             if (val!=1 || strcmp(buffer,"=")!=0) {
184                 status = ERROR_INPUT_PAR; break;
185             }
186             val = fscanf(fp,"%d",&ibuff);
187             if (val!=1) { status = ERROR_INPUT_PAR; break; }
188             Input->problem_num=ibuff;
189             fscanf(fp, "%*[^\n]"); // skip rest of line
190         }
191
192         else if (strcmp(buffer,"print_level")==0) {
```

```
193             val = fscanf(fp,"%s",buffer);
194             if (val!=1 || strcmp(buffer,"=")!=0) {
195                 status = ERROR_INPUT_PAR; break;
196             }
197             val = fscanf(fp,"%d",&ibuff);
198             if (val!=1) { status = ERROR_INPUT_PAR; break; }
199             Input->print_level = ibuff;
200             fscanf(fp, "%*[^\n]"); // skip rest of line
201         }
202 
203         else if (strcmp(buffer,"output_type")==0) {
204             val = fscanf(fp,"%s",buffer);
205             if (val!=1 || strcmp(buffer,"=")!=0) {
206                 status = ERROR_INPUT_PAR; break;
207             }
208             val = fscanf(fp,"%d",&ibuff);
209             if (val!=1) { status = ERROR_INPUT_PAR; break; }
210             Input->output_type = ibuff;
211             fscanf(fp, "%*[^\n]"); // skip rest of line
212         }
213 
214         else if (strcmp(buffer,"solver_type")==0) {
215             val = fscanf(fp,"%s",buffer);
216             if (val!=1 || strcmp(buffer,"=")!=0) {
217                 status = ERROR_INPUT_PAR; break;
218             }
219             val = fscanf(fp,"%d",&ibuff);
220             if (val!=1) { status = ERROR_INPUT_PAR; break; }
221             Input->solver_type = ibuff;
222             fscanf(fp, "%*[^\n]"); // skip rest of line
223         }
224 
225         else if (strcmp(buffer,"precond_type")==0) {
226             val = fscanf(fp,"%s",buffer);
227             if (val!=1 || strcmp(buffer,"=")!=0) {
228                 status = ERROR_INPUT_PAR; break;
229             }
230             val = fscanf(fp,"%d",&ibuff);
231             if (val!=1) { status = ERROR_INPUT_PAR; break; }
232             Input->precond_type = ibuff;
233             fscanf(fp, "%*[^\n]"); // skip rest of line
234         }
235 
236         else if (strcmp(buffer,"stop_type")==0) {
237             val = fscanf(fp,"%s",buffer);
238             if (val!=1 || strcmp(buffer,"=")!=0) {
239                 status = ERROR_INPUT_PAR; break;
240             }
241             val = fscanf(fp,"%d",&ibuff);
242             if (val!=1) { status = ERROR_INPUT_PAR; break; }
243             Input->stop_type = ibuff;
244             fscanf(fp, "%*[^\n]"); // skip rest of line
245         }
246 
247         else if (strcmp(buffer,"itsolver_tol")==0) {
248             val = fscanf(fp,"%s",buffer);
249             if (val!=1 || strcmp(buffer,"=")!=0) {
250                 status = ERROR_INPUT_PAR; break;
251             }
252             val = fscanf(fp,"%lf",&dbuff);
253             if (val!=1) { status = ERROR_INPUT_PAR; break; }
254             Input->itsolver_tol = dbuff;
255             fscanf(fp, "%*[^\n]"); // skip rest of line
256         }
257 
258         else if (strcmp(buffer,"itsolver_maxit")==0) {
259             val = fscanf(fp,"%s",buffer);
260             if (val!=1 || strcmp(buffer,"=")!=0) {
261                 status = ERROR_INPUT_PAR; break;
262             }
263             val = fscanf(fp,"%d",&ibuff);
264             if (val!=1) { status = ERROR_INPUT_PAR; break; }
265             Input->itsolver_maxit = ibuff;
266             fscanf(fp, "%*[^\n]"); // skip rest of line
267         }
268 
269         else if (strcmp(buffer,"solver_type_v")==0) {
270             val = fscanf(fp,"%s",buffer);
271             if (val!=1 || strcmp(buffer,"=")!=0) {
272                 status = ERROR_INPUT_PAR; break;
273             }
274             val = fscanf(fp,"%d",&ibuff);
275             if (val!=1) { status = ERROR_INPUT_PAR; break; }
276             Input->itsolver_type_v = ibuff;
277             fscanf(fp, "%*[^\n]"); // skip rest of line
278         }
279 
```

```
280         else if (strcmp(buffer,"precond_type_v")==0) {
281             val = fscanf(fp,"%s",buffer);
282             if (val!=1 || strcmp(buffer,"=")!=0) {
283                 status = ERROR_INPUT_PAR; break;
284             }
285             val = fscanf(fp,"%d",&ibuff);
286             if (val!=1) { status = ERROR_INPUT_PAR; break; }
287             Input->precond_type_v = ibuff;
288             fscanf(fp, "%*[^\n]"); // skip rest of line
289         }
290 
291         else if (strcmp(buffer,"itsolver_tol_v")==0) {
292             val = fscanf(fp,"%s",buffer);
293             if (val!=1 || strcmp(buffer,"=")!=0) {
294                 status = ERROR_INPUT_PAR; break;
295             }
296             val = fscanf(fp,"%lf",&dbuff);
297             if (val!=1) { status = ERROR_INPUT_PAR; break; }
298             Input->pre_tol_v = dbuff;
299             fscanf(fp, "%*[^\n]"); // skip rest of line
300         }
301 
302         else if (strcmp(buffer,"itsolver_maxit_v")==0) {
303             val = fscanf(fp,"%s",buffer);
304             if (val!=1 || strcmp(buffer,"=")!=0) {
305                 status = ERROR_INPUT_PAR; break;
306             }
307             val = fscanf(fp,"%d",&ibuff);
308             if (val!=1) { status = ERROR_INPUT_PAR; break; }
309             Input->pre_maxit_v = ibuff;
310             fscanf(fp, "%*[^\n]"); // skip rest of line
311         }
312 
313         else if (strcmp(buffer,"itsolver_restart_v")==0) {
314             val = fscanf(fp,"%s",buffer);
315             if (val!=1 || strcmp(buffer,"=")!=0) {
316                 status = ERROR_INPUT_PAR; break;
317             }
318             val = fscanf(fp,"%d",&ibuff);
319             if (val!=1) { status = ERROR_INPUT_PAR; break; }
320             Input->pre_restart_v = ibuff;
321             fscanf(fp, "%*[^\n]"); // skip rest of line
322         }
323 
324         else if (strcmp(buffer,"solver_type_p")==0) {
325             val = fscanf(fp,"%s",buffer);
326             if (val!=1 || strcmp(buffer,"=")!=0) {
327                 status = ERROR_INPUT_PAR; break;
328             }
329             val = fscanf(fp,"%d",&ibuff);
330             if (val!=1) { status = ERROR_INPUT_PAR; break; }
331             Input->itsolver_type_p = ibuff;
332             fscanf(fp, "%*[^\n]"); // skip rest of line
333         }
334 
335         else if (strcmp(buffer,"precond_type_p")==0) {
336             val = fscanf(fp,"%s",buffer);
337             if (val!=1 || strcmp(buffer,"=")!=0) {
338                 status = ERROR_INPUT_PAR; break;
339             }
340             val = fscanf(fp,"%d",&ibuff);
341             if (val!=1) { status = ERROR_INPUT_PAR; break; }
342             Input->precond_type_p = ibuff;
343             fscanf(fp, "%*[^\n]"); // skip rest of line
344         }
345 
346         else if (strcmp(buffer,"itsolver_tol_p")==0) {
347             val = fscanf(fp,"%s",buffer);
348             if (val!=1 || strcmp(buffer,"=")!=0) {
349                 status = ERROR_INPUT_PAR; break;
350             }
351             val = fscanf(fp,"%lf",&dbuff);
352             if (val!=1) { status = ERROR_INPUT_PAR; break; }
353             Input->pre_tol_p = dbuff;
354             fscanf(fp, "%*[^\n]"); // skip rest of line
355         }
356 
357         else if (strcmp(buffer,"itsolver_maxit_p")==0) {
358             val = fscanf(fp,"%s",buffer);
359             if (val!=1 || strcmp(buffer,"=")!=0) {
360                 status = ERROR_INPUT_PAR; break;
361             }
362             val = fscanf(fp,"%d",&ibuff);
363             if (val!=1) { status = ERROR_INPUT_PAR; break; }
364             Input->pre_maxit_p = ibuff;
365             fscanf(fp, "%*[^\n]"); // skip rest of line
366         }
```

```
367
368         else if (strcmp(buffer,"itsolver_restart_p")==0) {
369             val = fscanf(fp,"%s",buffer);
370             if (val!=1 || strcmp(buffer,"=")!=0) {
371                 status = ERROR_INPUT_PAR; break;
372             }
373             val = fscanf(fp,"%d",&ibuff);
374             if (val!=1) { status = ERROR_INPUT_PAR; break; }
375             Input->pre_restart_p = ibuff;
376             fscanf(fp, "%*[^\n]"); // skip rest of line
377         }
378
379         else if (strcmp(buffer,"itsolver_restart")==0) {
380             val = fscanf(fp,"%s",buffer);
381             if (val!=1 || strcmp(buffer,"=")!=0) {
382                 status = ERROR_INPUT_PAR; break;
383             }
384             val = fscanf(fp,"%d",&ibuff);
385             if (val!=1) { status = ERROR_INPUT_PAR; break; }
386             Input->restart = ibuff;
387             fscanf(fp, "%*[^\n]"); // skip rest of line
388         }
389
390         else if (strcmp(buffer,"AMG_ILU_levels_v")==0) {
391             val = fscanf(fp,"%s",buffer);
392             if (val!=1 || strcmp(buffer,"=")!=0) {
393                 status = ERROR_INPUT_PAR; break;
394             }
395             val = fscanf(fp,"%d",&ibuff);
396             if (val!=1) { status = ERROR_INPUT_PAR; break; }
397             Input->AMG_ILU_levels_v = ibuff;
398             fscanf(fp, "%*[^\n]"); // skip rest of line
399         }
400
401         else if (strcmp(buffer,"AMG_schwarz_levels_v")==0) {
402             val = fscanf(fp,"%s",buffer);
403             if (val!=1 || strcmp(buffer,"=")!=0) {
404                 status = ERROR_INPUT_PAR; break;
405             }
406             val = fscanf(fp,"%d",&ibuff);
407             if (val!=1) { status = FASP_SUCCESS; break; }
408             Input->AMG_schwarz_levels_v = ibuff;
409             fscanf(fp, "%*[^\n]"); // skip rest of line
410         }
411
412         else if (strcmp(buffer,"AMG_type_v")==0) {
413             val = fscanf(fp,"%s",buffer);
414             if (val!=1 || strcmp(buffer,"=")!=0) {
415                 status = ERROR_INPUT_PAR; break;
416             }
417             val = fscanf(fp,"%s",buffer);
418             if (val!=1) { status = ERROR_INPUT_PAR; break; }
419
420             if ((strcmp(buffer,"C")==0)||(strcmp(buffer,"c")==0))
421             Input->AMG_type_v = CLASSIC_AMG;
422             else if ((strcmp(buffer,"SA")==0)||(strcmp(buffer,"sa")==0))
423             Input->AMG_type_v = SA_AMG;
424             else if ((strcmp(buffer,"UA")==0)||(strcmp(buffer,"ua")==0))
425             Input->AMG_type_v = UA_AMG;
426             else
427             { status = ERROR_INPUT_PAR; break; }
428             fscanf(fp, "%*[^\n]"); // skip rest of line
429         }
430
431         else if (strcmp(buffer,"AMG_aggregation_type_v")==0) {
432             val = fscanf(fp,"%s",buffer);
433             if (val!=1 || strcmp(buffer,"=")!=0) {
434                 status = ERROR_INPUT_PAR; break;
435             }
436             val = fscanf(fp,"%d",&ibuff);
437             if (val!=1) { status = ERROR_INPUT_PAR; break; }
438             Input->AMG_aggregation_type_v = ibuff;
439             fscanf(fp, "%*[^\n]"); // skip rest of line
440         }
441
442         else if (strcmp(buffer,"AMG_pair_number_v")==0) {
443             val = fscanf(fp,"%s",buffer);
444             if (val!=1 || strcmp(buffer,"=")!=0) {
445                 status = ERROR_INPUT_PAR; break;
446             }
447             val = fscanf(fp,"%d",&ibuff);
448             if (val!=1) { status = ERROR_INPUT_PAR; break; }
449             Input->AMG_pair_number_v = ibuff;
450             fscanf(fp, "%*[^\n]"); // skip rest of line
451         }
452
453         else if (strcmp(buffer,"AMG_quality_bound_v")==0) {
```

```
454             val = fscanf(fp,"%s",buffer);
455             if (val!=1 || strcmp(buffer,"=")!=0) {
456                 status = ERROR_INPUT_PAR; break;
457             }
458             val = fscanf(fp,"%lf",&dbuff);
459             if (val!=1) { status = ERROR_INPUT_PAR; break; }
460             Input->AMG_quality_bound_v = dbuff;
461             fscanf(fp, "%*[^\n]"); // skip rest of line
462         }
463 
464         else if (strcmp(buffer,"AMG_strong_coupled_v")==0) {
465             val = fscanf(fp,"%s",buffer);
466             if (val!=1 || strcmp(buffer,"=")!=0) {
467                 status = ERROR_INPUT_PAR; break;
468             }
469             val = fscanf(fp,"%lf",&dbuff);
470             if (val!=1) { status = ERROR_INPUT_PAR; break; }
471             Input->AMG_strong_coupled_v = dbuff;
472             fscanf(fp, "%*[^\n]"); // skip rest of line
473         }
474 
475         else if (strcmp(buffer,"AMG_max_aggregation_v")==0) {
476             val = fscanf(fp,"%s",buffer);
477             if (val!=1 || strcmp(buffer,"=")!=0) {
478                 status = ERROR_INPUT_PAR; break;
479             }
480             val = fscanf(fp,"%d",&ibuff);
481             if (val!=1) { status = ERROR_INPUT_PAR; break; }
482             Input->AMG_max_aggregation_v = ibuff;
483             fscanf(fp, "%*[^\n]"); // skip rest of line
484         }
485 
486         else if (strcmp(buffer,"AMG_tentative_smooth_v")==0) {
487             val = fscanf(fp,"%s",buffer);
488             if (val!=1 || strcmp(buffer,"=")!=0) {
489                 status = ERROR_INPUT_PAR; break;
490             }
491             val = fscanf(fp,"%lf",&dbuff);
492             if (val!=1) { status = ERROR_INPUT_PAR; break; }
493             Input->AMG_tentative_smooth_v = dbuff;
494             fscanf(fp, "%*[^\n]"); // skip rest of line
495         }
496 
497         else if (strcmp(buffer,"AMG_smooth_filter_v")==0) {
498             val = fscanf(fp,"%s",buffer);
499             if (val!=1 || strcmp(buffer,"=")!=0) {
500                 status = ERROR_INPUT_PAR; break;
501             }
502             val = fscanf(fp,"%s",buffer);
503             if (val!=1) { status = ERROR_INPUT_PAR; break; }
504 
505             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
506                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
507             Input->AMG_smooth_filter_v = ON;
508             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
509                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
510                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
511                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
512             Input->AMG_smooth_filter_v = OFF;
513             else
514             { status = ERROR_INPUT_PAR; break; }
515             fscanf(fp, "%*[^\n]"); // skip rest of line
516         }
517 
518         else if (strcmp(buffer,"AMG_coarse_scaling_v")==0) {
519             val = fscanf(fp,"%s",buffer);
520             if (val!=1 || strcmp(buffer,"=")!=0) {
521                 status = ERROR_INPUT_PAR; break;
522             }
523             val = fscanf(fp,"%s",buffer);
524             if (val!=1) { status = ERROR_INPUT_PAR; break; }
525 
526             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
527                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
528             Input->AMG_coarse_scaling_v = ON;
529             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
530                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
531                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
532                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
533             Input->AMG_coarse_scaling_v = OFF;
534             else
535             { status = ERROR_INPUT_PAR; break; }
536             fscanf(fp, "%*[^\n]"); // skip rest of line
537         }
538 
539         else if (strcmp(buffer,"AMG_levels_v")==0) {
540             val = fscanf(fp,"%s",buffer);
```

```
541             if (val!=1 || strcmp(buffer,"=")!=0) {
542                 status = ERROR_INPUT_PAR; break;
543             }
544             val = fscanf(fp,"%d",&ibuff);
545             if (val!=1) { status = ERROR_INPUT_PAR; break; }
546             Input->AMG_levels_v = ibuff;
547             fscanf(fp, "%*[^\n]"); // skip rest of line
548         }
549 
550         else if (strcmp(buffer,"AMG_tol_v")==0) {
551             val = fscanf(fp,"%s",buffer);
552             if (val!=1 || strcmp(buffer,"=")!=0) {
553                 status = ERROR_INPUT_PAR; break;
554             }
555             val = fscanf(fp,"%lf",&dbuff);
556             if (val!=1) { status = ERROR_INPUT_PAR; break; }
557             Input->AMG_tol_v = dbuff;
558             fscanf(fp, "%*[^\n]"); // skip rest of line
559         }
560 
561         else if (strcmp(buffer,"AMG_maxit_v")==0) {
562             val = fscanf(fp,"%s",buffer);
563             if (val!=1 || strcmp(buffer,"=")!=0) {
564                 status = ERROR_INPUT_PAR; break;
565             }
566             val = fscanf(fp,"%d",&ibuff);
567             if (val!=1) { status = ERROR_INPUT_PAR; break; }
568             Input->AMG_maxit_v = ibuff;
569             fscanf(fp, "%*[^\n]"); // skip rest of line
570         }
571 
572         else if (strcmp(buffer,"AMG_coarse_dof_v")==0) {
573             val = fscanf(fp,"%s",buffer);
574             if (val!=1 || strcmp(buffer,"=")!=0) {
575                 status = ERROR_INPUT_PAR; break;
576             }
577             val = fscanf(fp,"%d",&ibuff);
578             if (val!=1) { status = ERROR_INPUT_PAR; break; }
579             Input->AMG_coarse_dof_v = ibuff;
580             fscanf(fp, "%*[^\n]"); // skip rest of line
581         }
582 
583         else if (strcmp(buffer,"AMG_coarse_solver_v")==0) {
584             val = fscanf(fp,"%s",buffer);
585             if (val!=1 || strcmp(buffer,"=")!=0) {
586                 status = ERROR_INPUT_PAR; break;
587             }
588             val = fscanf(fp,"%d",&ibuff);
589             if (val!=1) { status = ERROR_INPUT_PAR; break; }
590             Input->AMG_coarse_solver_v = ibuff;
591             fscanf(fp, "%*[^\n]"); // skip rest of line
592         }
593 
594         else if (strcmp(buffer,"AMG_cycle_type_v")==0) {
595             val = fscanf(fp,"%s",buffer);
596             if (val!=1 || strcmp(buffer,"=")!=0) {
597                 status = ERROR_INPUT_PAR; break;
598             }
599             val = fscanf(fp,"%s",buffer);
600             if (val!=1) { status = ERROR_INPUT_PAR; break; }
601 
602             if ((strcmp(buffer,"V")==0)||(strcmp(buffer,"v")==0))
603             Input->AMG_cycle_type_v = V_CYCLE;
604             else if ((strcmp(buffer,"W")==0)||(strcmp(buffer,"w")==0))
605             Input->AMG_cycle_type_v = W_CYCLE;
606             else if ((strcmp(buffer,"A")==0)||(strcmp(buffer,"a")==0))
607             Input->AMG_cycle_type_v = AMLI_CYCLE;
608             else if ((strcmp(buffer,"NA")==0)||(strcmp(buffer,"na")==0))
609             Input->AMG_cycle_type_v = NL_AMLI_CYCLE;
610             else
611             { status = ERROR_INPUT_PAR; break; }
612             fscanf(fp, "%*[^\n]"); // skip rest of line
613         }
614 
615         else if (strcmp(buffer,"AMG_smoother_v")==0) {
616             val = fscanf(fp,"%s",buffer);
617             if (val!=1 || strcmp(buffer,"=")!=0) {
618                 status = ERROR_INPUT_PAR; break;
619             }
620             val = fscanf(fp,"%s",buffer);
621             if (val!=1) { status = ERROR_INPUT_PAR; break; }
622 
623             if ((strcmp(buffer,"JACOBI")==0)||(strcmp(buffer,"jacobi")==0))
624             Input->AMG_smoother_v = SMOOTHER_JACOBI;
625             else if ((strcmp(buffer,"GS")==0)||(strcmp(buffer,"gs")==0))
626             Input->AMG_smoother_v = SMOOTHER_GS;
627             else if ((strcmp(buffer,"SGS")==0)||(strcmp(buffer,"sgs")==0))
```

```
628             Input->AMG_smoother_v = SMOOTHER_SGS;
629             else if ((strcmp(buffer,"CG")==0)||(strcmp(buffer,"cg")==0))
630             Input->AMG_smoother_v = SMOOTHER_CG;
631             else if ((strcmp(buffer,"SOR")==0)||(strcmp(buffer,"sor")==0))
632             Input->AMG_smoother_v = SMOOTHER_SOR;
633             else if ((strcmp(buffer,"SSOR")==0)||(strcmp(buffer,"ssor")==0))
634             Input->AMG_smoother_v = SMOOTHER_SSOR;
635             else if ((strcmp(buffer,"GSOR")==0)||(strcmp(buffer,"gsor")==0))
636             Input->AMG_smoother_v = SMOOTHER_GSOR;
637             else if ((strcmp(buffer,"SGSOR")==0)||(strcmp(buffer,"sgsor")==0))
638             Input->AMG_smoother_v = SMOOTHER_SGSOR;
639             else if ((strcmp(buffer,"POLY")==0)||(strcmp(buffer,"poly")==0))
640             Input->AMG_smoother_v = SMOOTHER_POLY;
641             else if ((strcmp(buffer,"L1_DIAG")==0)||(strcmp(buffer,"l1_diag")==0))
642             Input->AMG_smoother_v = SMOOTHER_L1DIAG;
643             else
644             { status = ERROR_INPUT_PAR; break; }
645             fscanf(fp, "%*[^\n]"); // skip rest of line
646         }
647
648         else if (strcmp(buffer,"AMG_smooth_order_v")==0) {
649             val = fscanf(fp,"%s",buffer);
650             if (val!=1 || strcmp(buffer,"=")!=0) {
651                 status = ERROR_INPUT_PAR; break;
652             }
653             val = fscanf(fp,"%s",buffer);
654             if (val!=1) { status = ERROR_INPUT_PAR; break; }
655
656             if ((strcmp(buffer,"NO")==0)||(strcmp(buffer,"no")==0))
657             Input->AMG_smooth_order_v = NO_ORDER;
658             else if ((strcmp(buffer,"CF")==0)||(strcmp(buffer,"cf")==0))
659             Input->AMG_smooth_order_v = CF_ORDER;
660             else
661             { status = ERROR_INPUT_PAR; break; }
662             fscanf(fp, "%*[^\n]"); // skip rest of line
663         }
664
665         else if (strcmp(buffer,"AMG_coarsening_type_v")==0) {
666             val = fscanf(fp,"%s",buffer);
667             if (val!=1 || strcmp(buffer,"=")!=0) {
668                 status = ERROR_INPUT_PAR; break;
669             }
670             val = fscanf(fp,"%d",&ibuff);
671             if (val!=1) { status = ERROR_INPUT_PAR; break; }
672             Input->AMG_coarsening_type_v = ibuff;
673             fscanf(fp, "%*[^\n]"); // skip rest of line
674         }
675
676         else if (strcmp(buffer,"AMG_interpolation_type_v")==0) {
677             val = fscanf(fp,"%s",buffer);
678             if (val!=1 || strcmp(buffer,"=")!=0) {
679                 status = ERROR_INPUT_PAR; break;
680             }
681             val = fscanf(fp,"%d",&ibuff);
682             if (val!=1) { status = ERROR_INPUT_PAR; break; }
683             Input->AMG_interpolation_type_v = ibuff;
684             fscanf(fp, "%*[^\n]"); // skip rest of line
685         }
686
687         else if (strcmp(buffer,"AMG_aggressive_level_v")==0) {
688             val = fscanf(fp,"%s",buffer);
689             if (val!=1 || strcmp(buffer,"=")!=0) {
690                 status = ERROR_INPUT_PAR; break;
691             }
692             val = fscanf(fp,"%d",&ibuff);
693             if (val!=1) { status = ERROR_INPUT_PAR; break; }
694             Input->AMG_aggressive_level_v = ibuff;
695             fscanf(fp, "%*[^\n]"); // skip rest of line
696         }
697
698         else if (strcmp(buffer,"AMG_aggressive_path_v")==0) {
699             val = fscanf(fp,"%s",buffer);
700             if (val!=1 || strcmp(buffer,"=")!=0) {
701                 status = ERROR_INPUT_PAR; break;
702             }
703             val = fscanf(fp,"%d",&ibuff);
704             if (val!=1) { status = ERROR_INPUT_PAR; break; }
705             Input->AMG_aggressive_path_v = ibuff;
706             fscanf(fp, "%*[^\n]"); // skip rest of line
707         }
708
709         else if (strcmp(buffer,"AMG_presmooth_iter_v")==0) {
710             val = fscanf(fp,"%s",buffer);
711             if (val!=1 || strcmp(buffer,"=")!=0) {
712                 status = ERROR_INPUT_PAR; break;
713             }
714             val = fscanf(fp,"%d",&ibuff);
```

```
715             if (val!=1) { status = ERROR_INPUT_PAR; break; }
716             Input->AMG_presmooth_iter_v = ibuff;
717             fscanf(fp, "%*[^\n]"); // skip rest of line
718         }
719
720         else if (strcmp(buffer,"AMG_postsmooth_iter_v")==0) {
721             val = fscanf(fp,"%s",buffer);
722             if (val!=1 || strcmp(buffer,"=")!=0) {
723                 status = ERROR_INPUT_PAR; break;
724             }
725             val = fscanf(fp,"%d",&ibuff);
726             if (val!=1) { status = ERROR_INPUT_PAR; break; }
727             Input->AMG_postsmooth_iter_v = ibuff;
728             fscanf(fp, "%*[^\n]"); // skip rest of line
729         }
730
731         else if (strcmp(buffer,"AMG_relaxation_v")==0) {
732             val = fscanf(fp,"%s",buffer);
733             if (val!=1 || strcmp(buffer,"=")!=0) {
734                 status = ERROR_INPUT_PAR; break;
735             }
736             val = fscanf(fp,"%lf",&dbuff);
737             if (val!=1) { status = ERROR_INPUT_PAR; break; }
738             Input->AMG_relaxation_v=dbuff;
739             fscanf(fp, "%*[^\n]"); // skip rest of line
740         }
741
742         else if (strcmp(buffer,"AMG_polynomial_degree_v")==0) {
743             val = fscanf(fp,"%s",buffer);
744             if (val!=1 || strcmp(buffer,"=")!=0) {
745                 status = ERROR_INPUT_PAR; break;
746             }
747             val = fscanf(fp,"%d",&ibuff);
748             if (val!=1) { status = ERROR_INPUT_PAR; break; }
749             Input->AMG_polynomial_degree_v = ibuff;
750             fscanf(fp, "%*[^\n]"); // skip rest of line
751         }
752
753         else if (strcmp(buffer,"AMG_strong_threshold_v")==0) {
754             val = fscanf(fp,"%s",buffer);
755             if (val!=1 || strcmp(buffer,"=")!=0) {
756                 status = ERROR_INPUT_PAR; break;
757             }
758             val = fscanf(fp,"%lf",&dbuff);
759             if (val!=1) { status = ERROR_INPUT_PAR; break; }
760             Input->AMG_strong_threshold_v = dbuff;
761             fscanf(fp, "%*[^\n]"); // skip rest of line
762         }
763
764         else if (strcmp(buffer,"AMG_truncation_threshold_v")==0) {
765             val = fscanf(fp,"%s",buffer);
766             if (val!=1 || strcmp(buffer,"=")!=0) {
767                 status = ERROR_INPUT_PAR; break;
768             }
769             val = fscanf(fp,"%lf",&dbuff);
770             if (val!=1) { status = ERROR_INPUT_PAR; break; }
771             Input->AMG_truncation_threshold_v = dbuff;
772             fscanf(fp, "%*[^\n]"); // skip rest of line
773         }
774
775         else if (strcmp(buffer,"AMG_max_row_sum_v")==0) {
776             val = fscanf(fp,"%s",buffer);
777             if (val!=1 || strcmp(buffer,"=")!=0) {
778                 status = ERROR_INPUT_PAR; break;
779             }
780             val = fscanf(fp,"%lf",&dbuff);
781             if (val!=1) { status = ERROR_INPUT_PAR; break; }
782             Input->AMG_max_row_sum_v = dbuff;
783             fscanf(fp, "%*[^\n]"); // skip rest of line
784         }
785
786         else if (strcmp(buffer,"AMG_amli_degree_v")==0) {
787             val = fscanf(fp,"%s",buffer);
788             if (val!=1 || strcmp(buffer,"=")!=0) {
789                 status = ERROR_INPUT_PAR; break;
790             }
791             val = fscanf(fp,"%d",&ibuff);
792             if (val!=1) { status = ERROR_INPUT_PAR; break; }
793             Input->AMG_amli_degree_v = ibuff;
794             fscanf(fp, "%*[^\n]"); // skip rest of line
795         }
796
797         else if (strcmp(buffer,"AMG_nl_amli_krylov_type_v")==0) {
798             val = fscanf(fp,"%s",buffer);
799             if (val!=1 || strcmp(buffer,"=")!=0) {
800                 status = ERROR_INPUT_PAR; break;
801             }
```

```
802             val = fscanf(fp,"%d",&ibuff);
803             if (val!=1) { status = ERROR_INPUT_PAR; break; }
804             Input->AMG_nl_amli_krylov_type_v = ibuff;
805             fscanf(fp, "%*[^\n]"); // skip rest of line
806         }
807
808         else if (strcmp(buffer,"AMG_ILU_levels_p")==0) {
809             val = fscanf(fp,"%s",buffer);
810             if (val!=1 || strcmp(buffer,"=")!=0) {
811                 status = ERROR_INPUT_PAR; break;
812             }
813             val = fscanf(fp,"%d",&ibuff);
814             if (val!=1) { status = ERROR_INPUT_PAR; break; }
815             Input->AMG_ILU_levels_p = ibuff;
816             fscanf(fp, "%*[^\n]"); // skip rest of line
817         }
818
819         else if (strcmp(buffer,"AMG_schwarz_levels_p")==0) {
820             val = fscanf(fp,"%s",buffer);
821             if (val!=1 || strcmp(buffer,"=")!=0) {
822                 status = ERROR_INPUT_PAR; break;
823             }
824             val = fscanf(fp,"%d",&ibuff);
825             if (val!=1) { status = FASP_SUCCESS; break; }
826             Input->AMG_schwarz_levels_p = ibuff;
827             fscanf(fp, "%*[^\n]"); // skip rest of line
828         }
829
830         else if (strcmp(buffer,"AMG_type_p")==0) {
831             val = fscanf(fp,"%s",buffer);
832             if (val!=1 || strcmp(buffer,"=")!=0) {
833                 status = ERROR_INPUT_PAR; break;
834             }
835             val = fscanf(fp,"%s",buffer);
836             if (val!=1) { status = ERROR_INPUT_PAR; break; }
837
838             if ((strcmp(buffer,"C")==0)||(strcmp(buffer,"c")==0))
839             Input->AMG_type_p = CLASSIC_AMG;
840             else if ((strcmp(buffer,"SA")==0)||(strcmp(buffer,"sa")==0))
841             Input->AMG_type_p = SA_AMG;
842             else if ((strcmp(buffer,"UA")==0)||(strcmp(buffer,"ua")==0))
843             Input->AMG_type_p = UA_AMG;
844             else
845             { status = ERROR_INPUT_PAR; break; }
846             fscanf(fp, "%*[^\n]"); // skip rest of line
847         }
848
849         else if (strcmp(buffer,"AMG_aggregation_type_p")==0) {
850             val = fscanf(fp,"%s",buffer);
851             if (val!=1 || strcmp(buffer,"=")!=0) {
852                 status = ERROR_INPUT_PAR; break;
853             }
854             val = fscanf(fp,"%d",&ibuff);
855             if (val!=1) { status = ERROR_INPUT_PAR; break; }
856             Input->AMG_aggregation_type_p = ibuff;
857             fscanf(fp, "%*[^\n]"); // skip rest of line
858         }
859
860         else if (strcmp(buffer,"AMG_pair_number_p")==0) {
861             val = fscanf(fp,"%s",buffer);
862             if (val!=1 || strcmp(buffer,"=")!=0) {
863                 status = ERROR_INPUT_PAR; break;
864             }
865             val = fscanf(fp,"%d",&ibuff);
866             if (val!=1) { status = ERROR_INPUT_PAR; break; }
867             Input->AMG_pair_number_p = ibuff;
868             fscanf(fp, "%*[^\n]"); // skip rest of line
869         }
870
871         else if (strcmp(buffer,"AMG_quality_bound_p")==0) {
872             val = fscanf(fp,"%s",buffer);
873             if (val!=1 || strcmp(buffer,"=")!=0) {
874                 status = ERROR_INPUT_PAR; break;
875             }
876             val = fscanf(fp,"%lf",&dbuff);
877             if (val!=1) { status = ERROR_INPUT_PAR; break; }
878             Input->AMG_quality_bound_p = dbuff;
879             fscanf(fp, "%*[^\n]"); // skip rest of line
880         }
881
882         else if (strcmp(buffer,"AMG_strong_coupled_p")==0) {
883             val = fscanf(fp,"%s",buffer);
884             if (val!=1 || strcmp(buffer,"=")!=0) {
885                 status = ERROR_INPUT_PAR; break;
886             }
887             val = fscanf(fp,"%lf",&dbuff);
888             if (val!=1) { status = ERROR_INPUT_PAR; break; }
```

```
889             Input->AMG_strong_coupled_p = dbuff;
890             fscanf(fp, "%*[^\n]"); // skip rest of line
891         }
892 
893         else if (strcmp(buffer,"AMG_max_aggregation_p")==0) {
894             val = fscanf(fp,"%s",buffer);
895             if (val!=1 || strcmp(buffer,"=")!=0) {
896                 status = ERROR_INPUT_PAR; break;
897             }
898             val = fscanf(fp,"%d",&ibuff);
899             if (val!=1) { status = ERROR_INPUT_PAR; break; }
900             Input->AMG_max_aggregation_p = ibuff;
901             fscanf(fp, "%*[^\n]"); // skip rest of line
902         }
903 
904         else if (strcmp(buffer,"AMG_tentative_smooth_p")==0) {
905             val = fscanf(fp,"%s",buffer);
906             if (val!=1 || strcmp(buffer,"=")!=0) {
907                 status = ERROR_INPUT_PAR; break;
908             }
909             val = fscanf(fp,"%lf",&dbuff);
910             if (val!=1) { status = ERROR_INPUT_PAR; break; }
911             Input->AMG_tentative_smooth_p = dbuff;
912             fscanf(fp, "%*[^\n]"); // skip rest of line
913         }
914 
915         else if (strcmp(buffer,"AMG_smooth_filter_p")==0) {
916             val = fscanf(fp,"%s",buffer);
917             if (val!=1 || strcmp(buffer,"=")!=0) {
918                 status = ERROR_INPUT_PAR; break;
919             }
920             val = fscanf(fp,"%s",buffer);
921             if (val!=1) { status = ERROR_INPUT_PAR; break; }
922 
923             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
924                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
925             Input->AMG_smooth_filter_p = ON;
926             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
927                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
928                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
929                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
930             Input->AMG_smooth_filter_p = OFF;
931             else
932             { status = ERROR_INPUT_PAR; break; }
933             fscanf(fp, "%*[^\n]"); // skip rest of line
934         }
935 
936         else if (strcmp(buffer,"AMG_coarse_scaling_p")==0) {
937             val = fscanf(fp,"%s",buffer);
938             if (val!=1 || strcmp(buffer,"=")!=0) {
939                 status = ERROR_INPUT_PAR; break;
940             }
941             val = fscanf(fp,"%s",buffer);
942             if (val!=1) { status = ERROR_INPUT_PAR; break; }
943 
944             if ((strcmp(buffer,"ON")==0)||(strcmp(buffer,"on")==0)||
945                 (strcmp(buffer,"On")==0)||(strcmp(buffer,"oN")==0))
946             Input->AMG_coarse_scaling_p = ON;
947             else if ((strcmp(buffer,"OFF")==0)||(strcmp(buffer,"off")==0)||
948                      (strcmp(buffer,"ofF")==0)||(strcmp(buffer,"oFf")==0)||
949                      (strcmp(buffer,"Off")==0)||(strcmp(buffer,"oFF")==0)||
950                      (strcmp(buffer,"OfF")==0)||(strcmp(buffer,"OFf")==0))
951             Input->AMG_coarse_scaling_p = OFF;
952             else
953             { status = ERROR_INPUT_PAR; break; }
954             fscanf(fp, "%*[^\n]"); // skip rest of line
955         }
956 
957         else if (strcmp(buffer,"AMG_levels_p")==0) {
958             val = fscanf(fp,"%s",buffer);
959             if (val!=1 || strcmp(buffer,"=")!=0) {
960                 status = ERROR_INPUT_PAR; break;
961             }
962             val = fscanf(fp,"%d",&ibuff);
963             if (val!=1) { status = ERROR_INPUT_PAR; break; }
964             Input->AMG_levels_p = ibuff;
965             fscanf(fp, "%*[^\n]"); // skip rest of line
966         }
967 
968         else if (strcmp(buffer,"AMG_tol_p")==0) {
969             val = fscanf(fp,"%s",buffer);
970             if (val!=1 || strcmp(buffer,"=")!=0) {
971                 status = ERROR_INPUT_PAR; break;
972             }
973             val = fscanf(fp,"%lf",&dbuff);
974             if (val!=1) { status = ERROR_INPUT_PAR; break; }
975             Input->AMG_tol_p = dbuff;
```

```
976             fscanf(fp, "%*[^\n]"); // skip rest of line
977         }
978
979         else if (strcmp(buffer,"AMG_maxit_p")==0) {
980             val = fscanf(fp,"%s",buffer);
981             if (val!=1 || strcmp(buffer,"=")!=0) {
982                 status = ERROR_INPUT_PAR; break;
983             }
984             val = fscanf(fp,"%d",&ibuff);
985             if (val!=1) { status = ERROR_INPUT_PAR; break; }
986             Input->AMG_maxit_p = ibuff;
987             fscanf(fp, "%*[^\n]"); // skip rest of line
988         }
989
990         else if (strcmp(buffer,"AMG_coarse_dof_p")==0) {
991             val = fscanf(fp,"%s",buffer);
992             if (val!=1 || strcmp(buffer,"=")!=0) {
993                 status = ERROR_INPUT_PAR; break;
994             }
995             val = fscanf(fp,"%d",&ibuff);
996             if (val!=1) { status = ERROR_INPUT_PAR; break; }
997             Input->AMG_coarse_dof_p = ibuff;
998             fscanf(fp, "%*[^\n]"); // skip rest of line
999         }
1000
1001         else if (strcmp(buffer,"AMG_coarse_solver_p")==0) {
1002             val = fscanf(fp,"%s",buffer);
1003             if (val!=1 || strcmp(buffer,"=")!=0) {
1004                 status = ERROR_INPUT_PAR; break;
1005             }
1006             val = fscanf(fp,"%d",&ibuff);
1007             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1008             Input->AMG_coarse_solver_p = ibuff;
1009             fscanf(fp, "%*[^\n]"); // skip rest of line
1010         }
1011
1012         else if (strcmp(buffer,"AMG_cycle_type_p")==0) {
1013             val = fscanf(fp,"%s",buffer);
1014             if (val!=1 || strcmp(buffer,"=")!=0) {
1015                 status = ERROR_INPUT_PAR; break;
1016             }
1017             val = fscanf(fp,"%s",buffer);
1018             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1019
1020             if ((strcmp(buffer,"V")==0)||(strcmp(buffer,"v")==0))
1021             Input->AMG_cycle_type_p = V_CYCLE;
1022             else if ((strcmp(buffer,"W")==0)||(strcmp(buffer,"w")==0))
1023             Input->AMG_cycle_type_p = W_CYCLE;
1024             else if ((strcmp(buffer,"A")==0)||(strcmp(buffer,"a")==0))
1025             Input->AMG_cycle_type_p = AMLI_CYCLE;
1026             else if ((strcmp(buffer,"NA")==0)||(strcmp(buffer,"na")==0))
1027             Input->AMG_cycle_type_p = NL_AMLI_CYCLE;
1028             else
1029             { status = ERROR_INPUT_PAR; break; }
1030             fscanf(fp, "%*[^\n]"); // skip rest of line
1031         }
1032
1033         else if (strcmp(buffer,"AMG_smoother_p")==0) {
1034             val = fscanf(fp,"%s",buffer);
1035             if (val!=1 || strcmp(buffer,"=")!=0) {
1036                 status = ERROR_INPUT_PAR; break;
1037             }
1038             val = fscanf(fp,"%s",buffer);
1039             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1040
1041             if ((strcmp(buffer,"JACOBI")==0)||(strcmp(buffer,"jacobi")==0))
1042             Input->AMG_smoother_p = SMOOTHER_JACOBI;
1043             else if ((strcmp(buffer,"GS")==0)||(strcmp(buffer,"gs")==0))
1044             Input->AMG_smoother_p = SMOOTHER_GS;
1045             else if ((strcmp(buffer,"SGS")==0)||(strcmp(buffer,"sgs")==0))
1046             Input->AMG_smoother_p = SMOOTHER_SGS;
1047             else if ((strcmp(buffer,"CG")==0)||(strcmp(buffer,"cg")==0))
1048             Input->AMG_smoother_p = SMOOTHER_CG;
1049             else if ((strcmp(buffer,"SOR")==0)||(strcmp(buffer,"sor")==0))
1050             Input->AMG_smoother_p = SMOOTHER_SOR;
1051             else if ((strcmp(buffer,"SSOR")==0)||(strcmp(buffer,"ssor")==0))
1052             Input->AMG_smoother_p = SMOOTHER_SSOR;
1053             else if ((strcmp(buffer,"GSOR")==0)||(strcmp(buffer,"gsor")==0))
1054             Input->AMG_smoother_p = SMOOTHER_GSOR;
1055             else if ((strcmp(buffer,"SGSOR")==0)||(strcmp(buffer,"sgsor")==0))
1056             Input->AMG_smoother_p = SMOOTHER_SGSOR;
1057             else if ((strcmp(buffer,"POLY")==0)||(strcmp(buffer,"poly")==0))
1058             Input->AMG_smoother_p = SMOOTHER_POLY;
1059             else if ((strcmp(buffer,"L1_DIAG")==0)||(strcmp(buffer,"l1_diag")==0))
1060             Input->AMG_smoother_p = SMOOTHER_L1DIAG;
1061             else
1062             { status = ERROR_INPUT_PAR; break; }
```

```
1063            fscanf(fp, "%*[^\n]"); // skip rest of line
1064        }
1065
1066        else if (strcmp(buffer,"AMG_smooth_order_p")==0) {
1067            val = fscanf(fp,"%s",buffer);
1068            if (val!=1 || strcmp(buffer,"=")!=0) {
1069                status = ERROR_INPUT_PAR; break;
1070            }
1071            val = fscanf(fp,"%s",buffer);
1072            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1073
1074            if ((strcmp(buffer,"NO")==0)||(strcmp(buffer,"no")==0))
1075            Input->AMG_smooth_order_p = NO_ORDER;
1076            else if ((strcmp(buffer,"CF")==0)||(strcmp(buffer,"cf")==0))
1077            Input->AMG_smooth_order_p = CF_ORDER;
1078            else
1079            { status = ERROR_INPUT_PAR; break; }
1080            fscanf(fp, "%*[^\n]"); // skip rest of line
1081        }
1082
1083        else if (strcmp(buffer,"AMG_coarsening_type_p")==0) {
1084            val = fscanf(fp,"%s",buffer);
1085            if (val!=1 || strcmp(buffer,"=")!=0) {
1086                status = ERROR_INPUT_PAR; break;
1087            }
1088            val = fscanf(fp,"%d",&ibuff);
1089            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1090            Input->AMG_coarsening_type_p = ibuff;
1091            fscanf(fp, "%*[^\n]"); // skip rest of line
1092        }
1093
1094        else if (strcmp(buffer,"AMG_interpolation_type_p")==0) {
1095            val = fscanf(fp,"%s",buffer);
1096            if (val!=1 || strcmp(buffer,"=")!=0) {
1097                status = ERROR_INPUT_PAR; break;
1098            }
1099            val = fscanf(fp,"%d",&ibuff);
1100            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1101            Input->AMG_interpolation_type_p = ibuff;
1102            fscanf(fp, "%*[^\n]"); // skip rest of line
1103        }
1104
1105        else if (strcmp(buffer,"AMG_aggressive_level_p")==0) {
1106            val = fscanf(fp,"%s",buffer);
1107            if (val!=1 || strcmp(buffer,"=")!=0) {
1108                status = ERROR_INPUT_PAR; break;
1109            }
1110            val = fscanf(fp,"%d",&ibuff);
1111            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1112            Input->AMG_aggressive_level_p = ibuff;
1113            fscanf(fp, "%*[^\n]"); // skip rest of line
1114        }
1115
1116        else if (strcmp(buffer,"AMG_aggressive_path_p")==0) {
1117            val = fscanf(fp,"%s",buffer);
1118            if (val!=1 || strcmp(buffer,"=")!=0) {
1119                status = ERROR_INPUT_PAR; break;
1120            }
1121            val = fscanf(fp,"%d",&ibuff);
1122            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1123            Input->AMG_aggressive_path_p = ibuff;
1124            fscanf(fp, "%*[^\n]"); // skip rest of line
1125        }
1126
1127        else if (strcmp(buffer,"AMG_presmooth_iter_p")==0) {
1128            val = fscanf(fp,"%s",buffer);
1129            if (val!=1 || strcmp(buffer,"=")!=0) {
1130                status = ERROR_INPUT_PAR; break;
1131            }
1132            val = fscanf(fp,"%d",&ibuff);
1133            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1134            Input->AMG_presmooth_iter_p = ibuff;
1135            fscanf(fp, "%*[^\n]"); // skip rest of line
1136        }
1137
1138        else if (strcmp(buffer,"AMG_postsmooth_iter_p")==0) {
1139            val = fscanf(fp,"%s",buffer);
1140            if (val!=1 || strcmp(buffer,"=")!=0) {
1141                status = ERROR_INPUT_PAR; break;
1142            }
1143            val = fscanf(fp,"%d",&ibuff);
1144            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1145            Input->AMG_postsmooth_iter_p = ibuff;
1146            fscanf(fp, "%*[^\n]"); // skip rest of line
1147        }
1148
1149        else if (strcmp(buffer,"AMG_relaxation_p")==0) {
```

```
1150            val = fscanf(fp,"%s",buffer);
1151            if (val!=1 || strcmp(buffer,"=")!=0) {
1152                status = ERROR_INPUT_PAR; break;
1153            }
1154            val = fscanf(fp,"%lf",&dbuff);
1155            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1156            Input->AMG_relaxation_p=dbuff;
1157            fscanf(fp, "%*[^\n]"); // skip rest of line
1158        }
1159
1160        else if (strcmp(buffer,"AMG_polynomial_degree_p")==0) {
1161            val = fscanf(fp,"%s",buffer);
1162            if (val!=1 || strcmp(buffer,"=")!=0) {
1163                status = ERROR_INPUT_PAR; break;
1164            }
1165            val = fscanf(fp,"%d",&ibuff);
1166            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1167            Input->AMG_polynomial_degree_p = ibuff;
1168            fscanf(fp, "%*[^\n]"); // skip rest of line
1169        }
1170
1171        else if (strcmp(buffer,"AMG_strong_threshold_p")==0) {
1172            val = fscanf(fp,"%s",buffer);
1173            if (val!=1 || strcmp(buffer,"=")!=0) {
1174                status = ERROR_INPUT_PAR; break;
1175            }
1176            val = fscanf(fp,"%lf",&dbuff);
1177            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1178            Input->AMG_strong_threshold_p = dbuff;
1179            fscanf(fp, "%*[^\n]"); // skip rest of line
1180        }
1181
1182        else if (strcmp(buffer,"AMG_truncation_threshold_p")==0) {
1183            val = fscanf(fp,"%s",buffer);
1184            if (val!=1 || strcmp(buffer,"=")!=0) {
1185                status = ERROR_INPUT_PAR; break;
1186            }
1187            val = fscanf(fp,"%lf",&dbuff);
1188            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1189            Input->AMG_truncation_threshold_p = dbuff;
1190            fscanf(fp, "%*[^\n]"); // skip rest of line
1191        }
1192
1193        else if (strcmp(buffer,"AMG_max_row_sum_p")==0) {
1194            val = fscanf(fp,"%s",buffer);
1195            if (val!=1 || strcmp(buffer,"=")!=0) {
1196                status = ERROR_INPUT_PAR; break;
1197            }
1198            val = fscanf(fp,"%lf",&dbuff);
1199            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1200            Input->AMG_max_row_sum_p = dbuff;
1201            fscanf(fp, "%*[^\n]"); // skip rest of line
1202        }
1203
1204        else if (strcmp(buffer,"AMG_amli_degree_p")==0) {
1205            val = fscanf(fp,"%s",buffer);
1206            if (val!=1 || strcmp(buffer,"=")!=0) {
1207                status = ERROR_INPUT_PAR; break;
1208            }
1209            val = fscanf(fp,"%d",&ibuff);
1210            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1211            Input->AMG_amli_degree_p = ibuff;
1212            fscanf(fp, "%*[^\n]"); // skip rest of line
1213        }
1214
1215        else if (strcmp(buffer,"AMG_nl_amli_krylov_type_p")==0) {
1216            val = fscanf(fp,"%s",buffer);
1217            if (val!=1 || strcmp(buffer,"=")!=0) {
1218                status = ERROR_INPUT_PAR; break;
1219            }
1220            val = fscanf(fp,"%d",&ibuff);
1221            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1222            Input->AMG_nl_amli_krylov_type_p = ibuff;
1223            fscanf(fp, "%*[^\n]"); // skip rest of line
1224        }
1225
1226        else if (strcmp(buffer,"ILU_type")==0) {
1227            val = fscanf(fp,"%s",buffer);
1228            if (val!=1 || strcmp(buffer,"=")!=0) {
1229                status = ERROR_INPUT_PAR; break;
1230            }
1231            val = fscanf(fp,"%d",&ibuff);
1232            if (val!=1) { status = ERROR_INPUT_PAR; break; }
1233            Input->ILU_type = ibuff;
1234            fscanf(fp, "%*[^\n]"); // skip rest of line
1235        }
1236
```

```
1237         else if (strcmp(buffer,"ILU_lfil")==0) {
1238             val = fscanf(fp,"%s",buffer);
1239             if (val!=1 || strcmp(buffer,"=")!=0) {
1240                 status = ERROR_INPUT_PAR; break;
1241             }
1242             val = fscanf(fp,"%d",&ibuff);
1243             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1244             Input->ILU_lfil = ibuff;
1245             fscanf(fp, "%*[^\n]"); // skip rest of line
1246         }
1247 
1248         else if (strcmp(buffer,"ILU_droptol")==0) {
1249             val = fscanf(fp,"%s",buffer);
1250             if (val!=1 || strcmp(buffer,"=")!=0) {
1251                 status = ERROR_INPUT_PAR; break;
1252             }
1253             val = fscanf(fp,"%lf",&dbuff);
1254             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1255             Input->ILU_droptol = dbuff;
1256             fscanf(fp, "%*[^\n]"); // skip rest of line
1257         }
1258 
1259         else if (strcmp(buffer,"ILU_relax")==0) {
1260             val = fscanf(fp,"%s",buffer);
1261             if (val!=1 || strcmp(buffer,"=")!=0) {
1262                 status = ERROR_INPUT_PAR; break;
1263             }
1264             val = fscanf(fp,"%lf",&dbuff);
1265             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1266             Input->ILU_relax = dbuff;
1267             fscanf(fp, "%*[^\n]"); // skip rest of line
1268         }
1269 
1270         else if (strcmp(buffer,"ILU_permtol")==0) {
1271             val = fscanf(fp,"%s",buffer);
1272             if (val!=1 || strcmp(buffer,"=")!=0) {
1273                 status = ERROR_INPUT_PAR; break;
1274             }
1275             val = fscanf(fp,"%lf",&dbuff);
1276             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1277             Input->ILU_permtol = dbuff;
1278             fscanf(fp, "%*[^\n]"); // skip rest of line
1279         }
1280 
1281         else if (strcmp(buffer,"SWZ_mmsize")==0) {
1282             val = fscanf(fp,"%s",buffer);
1283             if (val!=1 || strcmp(buffer,"=")!=0) {
1284                 status = ERROR_INPUT_PAR; break;
1285             }
1286             val = fscanf(fp,"%d",&ibuff);
1287             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1288             Input->SWZ_mmsize = ibuff;
1289             fscanf(fp, "%*[^\n]"); // skip rest of line
1290         }
1291 
1292         else if (strcmp(buffer,"SWZ_maxlvl")==0) {
1293             val = fscanf(fp,"%s",buffer);
1294             if (val!=1 || strcmp(buffer,"=")!=0) {
1295                 status = ERROR_INPUT_PAR; break;
1296             }
1297             val = fscanf(fp,"%d",&ibuff);
1298             if (val!=1) {status = ERROR_INPUT_PAR; break; }
1299             Input->SWZ_maxlvl = ibuff;
1300             fscanf(fp, "%*[^\n]"); // skip rest of line
1301         }
1302 
1303         else if (strcmp(buffer,"SWZ_type")==0) {
1304             val = fscanf(fp,"%s",buffer);
1305             if (val!=1 || strcmp(buffer,"=")!=0) {
1306                 status = ERROR_INPUT_PAR; break;
1307             }
1308             val = fscanf(fp,"%d",&ibuff);
1309             if (val!=1) { status = ERROR_INPUT_PAR; break; }
1310             Input->SWZ_type = ibuff;
1311             fscanf(fp, "%*[^\n]"); // skip rest of line
1312         }
1313 
1314         else {
1315             printf("### WARNING: Unknown input keyword %s!\n", buffer);
1316             fscanf(fp, "%*[^\n]"); // skip rest of line
1317         }
1318     }
1319 
1320     fclose(fp);
1321 
1322     // if meet unexpected input, stop the program
1323     fasp_chkerr(status, __FUNCTION__);
```

```
1324
1325     // sanity checks
1326     status = fasp_ns_param_check(Input);
1327
1328 #if DEBUG_MODE > 1
1329     printf("### DEBUG: Reading input status = %d\n", status);
1330 #endif
1331
1332     fasp_chkerr(status, __FUNCTION__);
1333 }
```

4.8.2.9 fasp_ns_param_solver_init()

```
void fasp_ns_param_solver_init (
             itsolver_ns_param * itsparam )
```

Initialize AMG parameters.

Parameters

amgparam	Parameters for AMG

Author

Lu Wang

Date

2014/02/11

Modified by Xiaozhe Hu on 02/21/2014

Definition at line 366 of file AuxParam.c.

```
367 {
368     itsparam->itsolver_type   = SOLVER_CG;
369     itsparam->precond_type    = PREC_AMG;
370     itsparam->stop_type       = STOP_REL_RES;
371     itsparam->maxit           = 100;
372     itsparam->tol             = 1e-8;
373     itsparam->restart         = 20;
374     itsparam->print_level     = 0;
375
376     // iterative solver parameter for the velocity block
377     itsparam->itsolver_type_v = SOLVER_CG;
378     itsparam->precond_type_v  = PREC_AMG;
379     itsparam->pre_maxit_v     = 20;
380     itsparam->pre_tol_v       = 1e-2;
381     itsparam->pre_restart_v   = 20;
382     itsparam->print_level_v   = 0;
383
384     // iterative solver parameter for the pressure block
385     itsparam->itsolver_type_p = SOLVER_CG;
386     itsparam->precond_type_p  = PREC_AMG;
387     itsparam->pre_maxit_p     = 20;
388     itsparam->pre_tol_p       = 1e-2;
389     itsparam->pre_restart_p   = 20;
390     itsparam->print_level_p   = 0;
391 }
```

4.8.2.10 fasp_ns_param_swz_set()

```
void fasp_ns_param_swz_set (
             SWZ_param * swzparam,
             input_ns_param * inparam )
```

Set SWZ_param with INPUT.

Parameters

swzparam	Parameters for Schwarz method
inparam	Input parameters

Author

Lu Wang

Date

2014/02/11

Definition at line 495 of file AuxParam.c.

```
497 {
498     swzparam->print_level = inparam->print_level;
499     swzparam->SWZ_type    = inparam->SWZ_type;
500     swzparam->SWZ_maxlvl  = inparam->SWZ_maxlvl;
501     swzparam->SWZ_mmsize  = inparam->SWZ_mmsize;
502 }
```

4.8.2.11 fasp_precond_ns_bdiag()

```
void fasp_precond_ns_bdiag (
             REAL * r,
             REAL * z,
             void * data )
```

block diagonal preconditioning for ns equation

Parameters

*r	pointer to residual
*z	pointer to preconditioned residual
*data	pointer to precondition data

Author

Xiaozhe Hu, Lu Wang

Date

10/20/2013

Note

modified by Lu Wang on 02/12/2014
Xiaozhe Hu modified on 02/21/2014
: modified by Xiaozhe Hu on May. 27, 2014

setup z;

Solve velocity

prepare AMG preconditioner

Solve Schur complement

Definition at line 41 of file PreNavierStokes.c.

```
44 {
45     precond_ns_data *predata=(precond_ns_data *)data;
46
47     const INT col = predata->col, colA = predata->colA, colB = predata->colB;
48
49     dvector rv; rv.row = colA; rv.val = r;
50     dvector zv; zv.row = colA; zv.val = z;
51     dvector rs; rs.row = colB; rs.val = r+colA;
52     dvector zs; zs.row = colB; zs.val = z+colA;
53
55     fasp_darray_set(col, z, 0.0);
56
57     //--------------------
59     //--------------------
61     AMG_data *mgl_v = predata->mgl_data_v;
62     AMG_param *amgparam_v = predata->param_v;
63     ITS_param *itparam_v = predata->ITS_param_v;
64
65 #if INEXACT
66
67     precond_data pcdata_v;
68     fasp_param_amg_to_prec(&pcdata_v, amgparam_v);
69     pcdata_v.max_levels = mgl_v[0].num_levels;
70     pcdata_v.mgl_data = predata->mgl_data_v;
71     precond pc_v; pc_v.data = &pcdata_v;
72     pc_v.fct = fasp_precond_amg;
73
74     if(itparam_v->print_level > 0) printf(COLOR_RESET "\n");
75     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
76
77 #else
78
79     dCSRmat tmpA;
80     dCSRmat *ptrA = &tmpA;
81     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
82     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
83     fasp_dcsr_free(ptrA);
84
85 #endif
86
87
88     //-------------------------
90     //-------------------------
91     ITS_param *itparam_p = predata->ITS_param_p;
92
93 #if INEXACT
94
95     if (itparam_p->precond_type == 1) {
96         precond pc_s;
97         pc_s.data = predata->diag_S;
98         pc_s.fct  = fasp_precond_diag;
99         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
100     }
101     else if (itparam_p->precond_type == 2){
```

```
103         AMG_data *mgl_p = predata->mgl_data_p;
104         AMG_param *amgparam_p = predata->param_p;
105 
106         precond_data pcdata_p;
107         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
108         pcdata_p.max_levels = mgl_p[0].num_levels;
109         pcdata_p.mgl_data = predata->mgl_data_p;
110         precond pc_p; pc_p.data = &pcdata_p;
111         pc_p.fct = fasp_precond_amg;
112 
113         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
114     }
115     else if (itparam_p->precond_type == 4) {
116 
117         ILU_data *LU_p = predata->ILU_p;
118 
119         precond pc_ilu;
120         pc_ilu.data = LU_p;
121         pc_ilu.fct  = fasp_precond_ilu;
122 
123         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
124 
125     }
126 
127 #else
128 
129     fasp_dcsr_trans(predata->S,ptrA);
130     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
131     fasp_dcsr_free(ptrA);
132 
133 #endif
134 
135     if(itparam_v->print_level > 0)
136         printf(COLOR_GREEN "\n");
137 
138 }
```

4.8.2.12 fasp_precond_ns_blu()

```
void fasp_precond_ns_blu (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute residule

Solve velocity

restore r

Definition at line 399 of file PreNavierStokes.c.

```
402 {
403
404     precond_ns_data *predata=(precond_ns_data *)data;
405     const int col = predata->col, colA = predata->colA, colB = predata->colB;
406
407     // local variables
408     double  *tempr = predata->w;
409
411     AMG_data *mgl_v = predata->mgl_data_v;
412     AMG_param *amgparam_v = predata->param_v;
413     ITS_param *itparam_v = predata->ITS_param_v;
414
415     dvector rv; rv.row = colA; rv.val = r;
416     dvector zv; zv.row = colA; zv.val = z;
417     dvector rs; rs.row = colB; rs.val = r+colA;
418     dvector zs; zs.row = colB; zs.val = z+colA;
419
421     fasp_darray_cp(col, r, tempr);
422     fasp_darray_set(col, z, 0.0);
423
424     //--------------------
426     //--------------------
427 #if INEXACT
428
429     precond_data pcdata_v;
430     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
431     pcdata_v.max_levels = mgl_v[0].num_levels;
432     pcdata_v.mgl_data = predata->mgl_data_v;
433     precond pc_v; pc_v.data = &pcdata_v;
434     pc_v.fct = fasp_precond_amg;
435
436     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
437
438     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
439 #else
440
441     dCSRmat tmpA;
442     dCSRmat *ptrA = &tmpA;
443     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
444     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
445     fasp_dcsr_free(ptrA);
446
447 #endif
448
449     //--------------------
451     //--------------------
452     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
453
454     //-------------------------
456     //-------------------------
457     ITS_param *itparam_p = predata->ITS_param_p;
458
459 #if INEXACT
460
461     if (itparam_p->precond_type == 1) {
462         precond pc_s;
463         pc_s.data = predata->diag_S;
464         pc_s.fct  = fasp_precond_diag;
465         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
466     }
467     else if (itparam_p->precond_type == 2){
469         AMG_data *mgl_p = predata->mgl_data_p;
470         AMG_param *amgparam_p = predata->param_p;
471
472         precond_data pcdata_p;
473         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
474         pcdata_p.max_levels = mgl_p[0].num_levels;
475         pcdata_p.mgl_data = predata->mgl_data_p;
476         precond pc_p; pc_p.data = &pcdata_p;
477         pc_p.fct = fasp_precond_amg;
478
479         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
480     }
481     else if (itparam_p->precond_type == 4) {
482
483         ILU_data *LU_p = predata->ILU_p;
484
485         precond pc_ilu;
486         pc_ilu.data = LU_p;
487         pc_ilu.fct  = fasp_precond_ilu;
488
489         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
490
```

```
491     }
492
493 #else
494
495     fasp_dcsr_trans(predata->S,ptrA);
496     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
497     fasp_dcsr_free(ptrA);
498
499 #endif
500
501     //--------------------
503     //--------------------
504     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
505
506     //--------------------
508     //--------------------
509 #if INEXACT
510
511     fasp_darray_set(colA, zv.val, 0.0);
512
513     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
514
515     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
516 #else
517
518     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
519     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
520     fasp_dcsr_free(ptrA);
521
522 #endif
523
524     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
526     fasp_darray_cp(col, tempr, r);
527
528 }
```

4.8.2.13 fasp_precond_ns_DGS()

```
void fasp_precond_ns_DGS (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement -BB^T

Compute zv = zv - B^T ∗ sp

Compute zs = zs + BB^T ∗ sp

restore r

Definition at line 1104 of file PreNavierStokes.c.

```
1107 {
1108     precond_ns_data *predata=(precond_ns_data *)data;
1109     const int col = predata->col, colA = predata->colA, colB = predata->colB;
1110 
1111     // local variables
1112     double  *tempr = predata->w;
1113 
1115     AMG_data *mgl_v = predata->mgl_data_v;
1116     AMG_param *amgparam_v = predata->param_v;
1117     ITS_param *itparam_v = predata->ITS_param_v;
1118 
1119     dvector rv; rv.row = colA; rv.val = r;
1120     dvector zv; zv.row = colA; zv.val = z;
1121     dvector rs; rs.row = colB; rs.val = r+colA;
1122     dvector zs; zs.row = colB; zs.val = z+colA;
1123 
1125     fasp_darray_cp(col, r, tempr);
1126     fasp_darray_set(col, z, 0.0);
1127 
1128     //--------------------
1130     //--------------------
1131 #if INEXACT
1132 
1133     precond_data pcdata_v;
1134     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1135     pcdata_v.max_levels = mgl_v[0].num_levels;
1136     pcdata_v.mgl_data = predata->mgl_data_v;
1137     precond pc_v; pc_v.data = &pcdata_v;
1138     pc_v.fct = fasp_precond_amg;
1139 
1140     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1141 
1142     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
1143 #else
1144 
1145     dCSRmat tmpA;
1146     dCSRmat *ptrA = &tmpA;
1147     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
1148     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1149     fasp_dcsr_free(ptrA);
1150 
1151 #endif
1152 
1153     //--------------------
1155     //--------------------
1156     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1157 
1158     //--------------------------
1160     //--------------------------
1161     ITS_param *itparam_p = predata->ITS_param_p;
1162 
1163 #if INEXACT
1164 
1165     if (itparam_p->precond_type == 1) {
1166         precond pc_s;
1167         pc_s.data = predata->diag_S;
1168         pc_s.fct  = fasp_precond_diag;
1169         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_s, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
1170     }
1171     else if (itparam_p->precond_type == 2){
1173         AMG_data *mgl_p = predata->mgl_data_p;
1174         AMG_param *amgparam_p = predata->param_p;
1175 
1176         precond_data pcdata_p;
1177         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1178         pcdata_p.max_levels = mgl_p[0].num_levels;
1179         pcdata_p.mgl_data = predata->mgl_data_p;
1180         precond pc_p; pc_p.data = &pcdata_p;
1181         pc_p.fct = fasp_precond_amg;
1182 
1183         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, predata->sp, &pc_p, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
1184     }
1185     else if (itparam_p->precond_type == 4) {
1186 
1187         ILU_data *LU_p = predata->ILU_p;
1188 
1189         precond pc_ilu;
1190         pc_ilu.data = LU_p;
1191         pc_ilu.fct  = fasp_precond_ilu;
1192 
1193         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_ilu, itparam_p->tol,itparam_p->maxit
    , itparam_p->restart, 1, itparam_p->print_level);
1194 
1195     }
```

```
1196
1197 #else
1198
1199     //dCSRmat tmpA;
1200     //dCSRmat *ptrA = &tmpA;
1201     fasp_dcsr_trans(predata->S,ptrA);
1202     fasp_solver_umfpack(ptrA, &rs, predata->sp, 0);
1203     fasp_dcsr_free(ptrA);
1204
1205 #endif
1206
1207     //--------------------
1209     //--------------------
1210     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, predata->sp->val, zv.val);
1211
1212     //--------------------
1214     //--------------------
1215     fasp_blas_dcsr_aAxpy(1.0, predata->S, predata->sp->val, zs.val);
1216
1217
1218     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1220     fasp_darray_cp(col, tempr, r);
1221 }
```

4.8.2.14 fasp_precond_ns_low_btri()

```
void fasp_precond_ns_low_btri (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

restore r

Definition at line 155 of file PreNavierStokes.c.

```
158 {
159     precond_ns_data *predata=(precond_ns_data *)data;
160     const int col = predata->col, colA = predata->colA, colB = predata->colB;
161
162     // local variables
163     double  *tempr = predata->w;
164
166     AMG_data *mgl_v = predata->mgl_data_v;
167     AMG_param *amgparam_v = predata->param_v;
168     ITS_param *itparam_v = predata->ITS_param_v;
169
170     dvector rv; rv.row = colA; rv.val = r;
171     dvector zv; zv.row = colA; zv.val = z;
172     dvector rs; rs.row = colB; rs.val = r+colA;
173     dvector zs; zs.row = colB; zs.val = z+colA;
174
176     fasp_darray_cp(col, r, tempr);
177     fasp_darray_set(col, z, 0.0);
178
179     //--------------------
181     //--------------------
182 #if INEXACT
183
184     precond_data pcdata_v;
```

```
185     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
186     pcdata_v.max_levels = mgl_v[0].num_levels;
187     pcdata_v.mgl_data = predata->mgl_data_v;
188     precond pc_v; pc_v.data = &pcdata_v;
189     pc_v.fct = fasp_precond_amg;
190 
191     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
192 
193     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
194 #else
195 
196     dCSRmat tmpA;
197     dCSRmat *ptrA = &tmpA;
198     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
199     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
200     fasp_dcsr_free(ptrA);
201 
202 #endif
203 
204     //-------------------
206     //-------------------
207     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
208 
209     //-------------------------
211     //-------------------------
212     ITS_param *itparam_p = predata->ITS_param_p;
213 
214 #if INEXACT
215 
216     if (itparam_p->precond_type == 1) {
217         precond pc_s;
218         pc_s.data = predata->diag_S;
219         pc_s.fct  = fasp_precond_diag;
220         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
221     }
222     else if (itparam_p->precond_type == 2){
224         AMG_data *mgl_p = predata->mgl_data_p;
225         AMG_param *amgparam_p = predata->param_p;
226 
227         precond_data pcdata_p;
228         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
229         pcdata_p.max_levels = mgl_p[0].num_levels;
230         pcdata_p.mgl_data = predata->mgl_data_p;
231         precond pc_p; pc_p.data = &pcdata_p;
232         pc_p.fct = fasp_precond_amg;
233 
234         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
235     }
236     else if (itparam_p->precond_type == 4) {
237 
238         ILU_data *LU_p = predata->ILU_p;
239 
240         precond pc_ilu;
241         pc_ilu.data = LU_p;
242         pc_ilu.fct  = fasp_precond_ilu;
243 
244         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
245 
246     }
247 
248 #else
249 
250     //dCSRmat tmpA;
251     //dCSRmat *ptrA = &tmpA;
252     fasp_dcsr_trans(predata->S,ptrA);
253     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
254     fasp_dcsr_free(ptrA);
255 
256 #endif
257 
258     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
260     fasp_darray_cp(col, tempr, r);
261 }
```

4.8.2.15 fasp_precond_ns_LSCDGS()

```
void fasp_precond_ns_LSCDGS (
```

```
            REAL * r,
            REAL * z,
            void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute zv = zv + B^T * sp

Compute rs = BAB^t * sp

Solve Schur complement

Compute zs = -zs

restore r

Definition at line 1235 of file PreNavierStokes.c.

```
1238 {
1239     precond_ns_data *predata=(precond_ns_data *)data;
1240     const int col = predata->col, colA = predata->colA, colB = predata->colB;
1241
1242     // local variables
1243     double  *tempr = predata->w;
1244
1246     AMG_data *mgl_v = predata->mgl_data_v;
1247     AMG_param *amgparam_v = predata->param_v;
1248     ITS_param *itparam_v = predata->ITS_param_v;
1249
1250     dvector rv; rv.row = colA; rv.val = r;
1251     dvector zv; zv.row = colA; zv.val = z;
1252     dvector rs; rs.row = colB; rs.val = r+colA;
1253     dvector zs; zs.row = colB; zs.val = z+colA;
1254
1256     fasp_darray_cp(col, r, tempr);
1257     fasp_darray_set(col, z, 0.0);
1258
1259     //-------------------
1261     //-------------------
1262 #if INEXACT
1263
1264     precond_data pcdata_v;
1265     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1266     pcdata_v.max_levels = mgl_v[0].num_levels;
1267     pcdata_v.mgl_data = predata->mgl_data_v;
1268     precond pc_v; pc_v.data = &pcdata_v;
1269     pc_v.fct = fasp_precond_amg;
1270
1271     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1272
1273     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
1274 #else
1275
1276     dCSRmat tmpA;
1277     dCSRmat *ptrA = &tmpA;
1278     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
1279     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1280     fasp_dcsr_free(ptrA);
1281
1282 #endif
1283
1284     //-------------------
1286     //-------------------
1287     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1288
```

```
1289     //--------------------------
1291     //--------------------------
1292     ITS_param *itparam_p = predata->ITS_param_p;
1293 
1294 #if INEXACT
1295 
1296     if (itparam_p->precond_type == 1) {
1297         precond pc_s;
1298         pc_s.data = predata->diag_S;
1299         pc_s.fct  = fasp_precond_diag;
1300         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_s, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
1301     }
1302     else if (itparam_p->precond_type == 2){
1304         AMG_data *mgl_p = predata->mgl_data_p;
1305         AMG_param *amgparam_p = predata->param_p;
1306 
1307         precond_data pcdata_p;
1308         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1309         pcdata_p.max_levels = mgl_p[0].num_levels;
1310         pcdata_p.mgl_data = predata->mgl_data_p;
1311         precond pc_p; pc_p.data = &pcdata_p;
1312         pc_p.fct = fasp_precond_amg;
1313 
1314         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, predata->sp, &pc_p, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
1315     }
1316     else if (itparam_p->precond_type == 4) {
1317 
1318         ILU_data *LU_p = predata->ILU_p;
1319 
1320         precond pc_ilu;
1321         pc_ilu.data = LU_p;
1322         pc_ilu.fct  = fasp_precond_ilu;
1323 
1324         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_ilu, itparam_p->tol,itparam_p->maxit
      , itparam_p->restart, 1, itparam_p->print_level);
1325 
1326     }
1327 
1328     /*
1329      fasp_dcoo_write("Ap.dat",  predata->S);
1330      fasp_dvec_write("rp.dat", &rs);
1331      getchar();
1332      */
1333 
1334 #else
1335 
1336     //dCSRmat tmpA;
1337     //dCSRmat *ptrA = &tmpA;
1338     fasp_dcsr_trans(predata->S,ptrA);
1339     fasp_solver_umfpack(ptrA, &rs, predata->sp, 0);
1340     fasp_dcsr_free(ptrA);
1341 
1342 #endif
1343 
1344     // change the sign of the solution
1345     fasp_blas_darray_ax(predata->sp->row, -1.0, predata->sp->val);
1346 
1347     //--------------------
1349     //--------------------
1350     fasp_blas_dcsr_aAxpy(1.0, predata->Bt, predata->sp->val, zv.val);
1351 
1352     //--------------------
1354     //--------------------
1355     fasp_blas_dcsr_mxv(predata->BABt, predata->sp->val, rs.val);
1356 
1357     //--------------------------
1359     //--------------------------
1360 
1361 #if INEXACT
1362 
1363     if (itparam_p->precond_type == 1) {
1364         precond pc_s;
1365         pc_s.data = predata->diag_S;
1366         pc_s.fct  = fasp_precond_diag;
1367         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
1368     }
1369     else if (itparam_p->precond_type == 2){
1371         AMG_data *mgl_p = predata->mgl_data_p;
1372         AMG_param *amgparam_p = predata->param_p;
1373 
1374         precond_data pcdata_p;
1375         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1376         pcdata_p.max_levels = mgl_p[0].num_levels;
1377         pcdata_p.mgl_data = predata->mgl_data_p;
```

```
1378         precond pc_p; pc_p.data = &pcdata_p;
1379         pc_p.fct = fasp_precond_amg;
1380 
1381         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
     ->restart, 1, itparam_p->print_level);
1382     }
1383     else if (itparam_p->precond_type == 4) {
1384 
1385         ILU_data *LU_p = predata->ILU_p;
1386 
1387         precond pc_ilu;
1388         pc_ilu.data = LU_p;
1389         pc_ilu.fct  = fasp_precond_ilu;
1390 
1391         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
     itparam_p->restart, 1, itparam_p->print_level);
1392 
1393     }
1394 
1395 #else
1396 
1397     //dCSRmat tmpA;
1398     //dCSRmat *ptrA = &tmpA;
1399     fasp_dcsr_trans(predata->S,ptrA);
1400     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
1401     fasp_dcsr_free(ptrA);
1402 
1403 #endif
1404 
1405     //--------------------
1407     //--------------------
1408     //fasp_blas_darray_ax(colB,-1.0,zs.val);
1409 
1410 
1411     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1413     fasp_darray_cp(col, tempr, r);
1414 }
```

4.8.2.16 fasp_precond_ns_projection()

```
void fasp_precond_ns_projection (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement -B*B^T

Compute zv = zv - B^T * zs

Compute zs = sp

restore r

Definition at line 971 of file PreNavierStokes.c.

```
974 {
975     precond_ns_data *predata=(precond_ns_data *)data;
976     const int col = predata->col, colA = predata->colA, colB = predata->colB;
977 
978     // local variables
979     double  *tempr = predata->w;
980 
982     AMG_data *mgl_v = predata->mgl_data_v;
983     AMG_param *amgparam_v = predata->param_v;
984     ITS_param *itparam_v = predata->ITS_param_v;
985 
986     dvector rv; rv.row = colA; rv.val = r;
987     dvector zv; zv.row = colA; zv.val = z;
988     dvector rs; rs.row = colB; rs.val = r+colA;
989     dvector zs; zs.row = colB; zs.val = z+colA;
990 
992     fasp_darray_cp(col, r, tempr);
993     fasp_darray_set(col, z, 0.0);
994 
995     //--------------------
997     //--------------------
998 #if INEXACT
999 
1000     precond_data pcdata_v;
1001     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1002     pcdata_v.max_levels = mgl_v[0].num_levels;
1003     pcdata_v.mgl_data = predata->mgl_data_v;
1004     precond pc_v; pc_v.data = &pcdata_v;
1005     pc_v.fct = fasp_precond_amg;
1006 
1007     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1008 
1009     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
     restart, 1, itparam_v->print_level);
1010 #else
1011 
1012     dCSRmat tmpA;
1013     dCSRmat *ptrA = &tmpA;
1014     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
1015     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1016     fasp_dcsr_free(ptrA);
1017 
1018 #endif
1019 
1020     //--------------------
1022     //--------------------
1023     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1024     fasp_darray_cp(colB, rs.val, predata->sp->val);
1025 
1026     //-------------------------
1028     //-------------------------
1029     ITS_param *itparam_p = predata->ITS_param_p;
1030 
1031 #if INEXACT
1032 
1033     if (itparam_p->precond_type == 1) {
1034         precond pc_s;
1035         pc_s.data = predata->diag_S;
1036         pc_s.fct  = fasp_precond_diag;
1037         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
     ->restart, 1, itparam_p->print_level);
1038     }
1039     else if (itparam_p->precond_type == 2){
1041         AMG_data *mgl_p = predata->mgl_data_p;
1042         AMG_param *amgparam_p = predata->param_p;
1043 
1044         precond_data pcdata_p;
1045         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1046         pcdata_p.max_levels = mgl_p[0].num_levels;
1047         pcdata_p.mgl_data = predata->mgl_data_p;
1048         precond pc_p; pc_p.data = &pcdata_p;
1049         pc_p.fct = fasp_precond_amg;
1050 
1051         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
     ->restart, 1, itparam_p->print_level);
1052     }
1053     else if (itparam_p->precond_type == 4) {
1054 
1055         ILU_data *LU_p = predata->ILU_p;
1056 
1057         precond pc_ilu;
1058         pc_ilu.data = LU_p;
1059         pc_ilu.fct  = fasp_precond_ilu;
1060 
1061         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
     itparam_p->restart, 1, itparam_p->print_level);
1062 
```

```
1063     }
1064
1065 #else
1066
1067     //dCSRmat tmpA;
1068     //dCSRmat *ptrA = &tmpA;
1069     fasp_dcsr_trans(predata->S,ptrA);
1070     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
1071     fasp_dcsr_free(ptrA);
1072
1073 #endif
1074
1075     //-------------------
1077     //-------------------
1078     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, zv.val);
1079
1080     //-------------------
1082     //-------------------
1083     fasp_darray_cp(colB, predata->sp->val, zs.val);
1084
1085
1086     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1088     fasp_darray_cp(col, tempr, r);
1089 }
```

4.8.2.17 fasp_precond_ns_simple()

```
void fasp_precond_ns_simple (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute zu = zu - D^{-1}B^T zs

restore r

Definition at line 542 of file PreNavierStokes.c.

```
545 {
546
547     precond_ns_data *predata=(precond_ns_data *)data;
548     const int col = predata->col, colA = predata->colA, colB = predata->colB;
549
550     // local variables
551     double  *tempr = predata->w;
552     INT i;
553
555     AMG_data *mgl_v = predata->mgl_data_v;
556     AMG_param *amgparam_v = predata->param_v;
557     ITS_param *itparam_v = predata->ITS_param_v;
558
559     dvector rv; rv.row = colA; rv.val = r;
560     dvector zv; zv.row = colA; zv.val = z;
561     dvector rs; rs.row = colB; rs.val = r+colA;
562     dvector zs; zs.row = colB; zs.val = z+colA;
563
565     fasp_darray_cp(col, r, tempr);
566     fasp_darray_set(col, z, 0.0);
```

```
567
568     //--------------------
570     //--------------------
571 #if INEXACT
572
573     precond_data pcdata_v;
574     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
575     pcdata_v.max_levels = mgl_v[0].num_levels;
576     pcdata_v.mgl_data = predata->mgl_data_v;
577     precond pc_v; pc_v.data = &pcdata_v;
578     pc_v.fct = fasp_precond_amg;
579
580     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
581
582     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
583 #else
584
585     dCSRmat tmpA;
586     dCSRmat *ptrA = &tmpA;
587     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
588     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
589     fasp_dcsr_free(ptrA);
590
591 #endif
592
593     //--------------------
595     //--------------------
596     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
597
598     //--------------------------
600     //--------------------------
601     ITS_param *itparam_p = predata->ITS_param_p;
602
603 #if INEXACT
604
605     if (itparam_p->precond_type == 1) {
606         precond pc_s;
607         pc_s.data = predata->diag_S;
608         pc_s.fct  = fasp_precond_diag;
609         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
610     }
611     else if (itparam_p->precond_type == 2){
613         AMG_data *mgl_p = predata->mgl_data_p;
614         AMG_param *amgparam_p = predata->param_p;
615
616         precond_data pcdata_p;
617         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
618         pcdata_p.max_levels = mgl_p[0].num_levels;
619         pcdata_p.mgl_data = predata->mgl_data_p;
620         precond pc_p; pc_p.data = &pcdata_p;
621         pc_p.fct = fasp_precond_amg;
622
623         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
624     }
625     else if (itparam_p->precond_type == 4) {
626
627         ILU_data *LU_p = predata->ILU_p;
628
629         precond pc_ilu;
630         pc_ilu.data = LU_p;
631         pc_ilu.fct  = fasp_precond_ilu;
632
633         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
634
635     }
636
637 #else
638
639     fasp_dcsr_trans(predata->S,ptrA);
640     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
641     fasp_dcsr_free(ptrA);
642
643 #endif
644
645     //--------------------
647     //--------------------
648     fasp_blas_dcsr_mxv(predata->Bt, zs.val, rv.val); // rv = B^T zs
649
650     for (i=0;i<colA;i++)
651     {
652         if (predata->diag_A->val[i] > SMALLREAL) rv.val[i]   = rv.val[i]/predata->
      diag_A->val[i]; // rv = D^{-1}rv
653     }
```

```
654
655     fasp_blas_darray_axpy (colA, -1.0, rv.val, zv.val); // zu = zu - rv
656
657
658     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
659     //-------------------
661     //-------------------
662     fasp_darray_cp(col, tempr, r);
663
664 }
```

4.8.2.18 fasp_precond_ns_simpler()

```
void fasp_precond_ns_simpler (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Compute rs = rs - B D^{-1} rv

Solve Schur complement

Compute residule

Solve velocity

restore r

Compute residule

Solve Schur complement

Compute zs = zs + deltaS

Compute zu = zu - D^{-1}B^T deltaS

restore r

Definition at line 678 of file PreNavierStokes.c.

```
681 {
682
683     precond_ns_data *predata=(precond_ns_data *)data;
684     const int col = predata->col, colA = predata->colA, colB = predata->colB;
685
686     // local variables
687     double  *tempr = predata->w;
688     INT i;
689
690     dvector *deltaS = predata->sp;
691
693     AMG_data *mgl_v = predata->mgl_data_v;
694     AMG_param *amgparam_v = predata->param_v;
695     ITS_param *itparam_v = predata->ITS_param_v;
696
697     dvector rv; rv.row = colA; rv.val = r;
698     dvector zv; zv.row = colA; zv.val = z;
699     dvector rs; rs.row = colB; rs.val = r+colA;
700     dvector zs; zs.row = colB; zs.val = z+colA;
```

```
701
703     fasp_darray_cp(col, r, tempr);
704     fasp_darray_set(col, z, 0.0);
705
706     //-------------------
708     //-------------------
709     for (i=0;i<colA;i++)
710     {
711         if (predata->diag_A->val[i] > SMALLREAL) zv.val[i]   = rv.val[i]/predata->
    diag_A->val[i]; // zv = D^{-1}rv
712     }
713
714     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val); // rs = rs - B zv
715
716     //-------------------------
718     //-------------------------
719     ITS_param *itparam_p = predata->ITS_param_p;
720
721 #if INEXACT
722
723     if (itparam_p->precond_type == 1) {
724         precond pc_s;
725         pc_s.data = predata->diag_S;
726         pc_s.fct  = fasp_precond_diag;
727         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
728     }
729     else if (itparam_p->precond_type == 2){
731         AMG_data *mgl_p = predata->mgl_data_p;
732         AMG_param *amgparam_p = predata->param_p;
733
734         precond_data pcdata_p;
735         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
736         pcdata_p.max_levels = mgl_p[0].num_levels;
737         pcdata_p.mgl_data = predata->mgl_data_p;
738         precond pc_p; pc_p.data = &pcdata_p;
739         pc_p.fct = fasp_precond_amg;
740
741         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
742     }
743     else if (itparam_p->precond_type == 4) {
744
745         ILU_data *LU_p = predata->ILU_p;
746
747         precond pc_ilu;
748         pc_ilu.data = LU_p;
749         pc_ilu.fct  = fasp_precond_ilu;
750
751         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
752
753     }
754
755 #else
756
757     dCSRmat tmpA;
758     dCSRmat *ptrA = &tmpA;
759     fasp_dcsr_trans(predata->S,ptrA);
760     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
761     fasp_dcsr_free(ptrA);
762
763 #endif
764
765     //-------------------
767     //-------------------
768     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
769
770     //-------------------
772     //-------------------
773 #if INEXACT
774
775     precond_data pcdata_v;
776     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
777     pcdata_v.max_levels = mgl_v[0].num_levels;
778     pcdata_v.mgl_data = predata->mgl_data_v;
779     precond pc_v; pc_v.data = &pcdata_v;
780     pc_v.fct = fasp_precond_amg;
781
782     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
783
784     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
785 #else
786
787     //dCSRmat tmpA;
788     //dCSRmat *ptrA = &tmpA;
```

```
789     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
790     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
791     fasp_dcsr_free(ptrA);
792
793 #endif
794
795
796     //-------------------
798     //-------------------
799     fasp_darray_cp(col, tempr, r);
800
801     //-------------------
803     //-------------------
804     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
805
806     //-------------------------
808     //-------------------------
809
810 #if INEXACT
811
812     if (itparam_p->precond_type == 1) {
813         precond pc_s;
814         pc_s.data = predata->diag_S;
815         pc_s.fct  = fasp_precond_diag;
816         fasp_solver_dcsr_pvfgmres(predata->S, &rs, deltaS, &pc_s, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
817     }
818     else if (itparam_p->precond_type == 2){
820         AMG_data *mgl_p = predata->mgl_data_p;
821         AMG_param *amgparam_p = predata->param_p;
822
823         precond_data pcdata_p;
824         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
825         pcdata_p.max_levels = mgl_p[0].num_levels;
826         pcdata_p.mgl_data = predata->mgl_data_p;
827         precond pc_p; pc_p.data = &pcdata_p;
828         pc_p.fct = fasp_precond_amg;
829
830         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, deltaS, &pc_p, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
831     }
832     else if (itparam_p->precond_type == 4) {
833
834         ILU_data *LU_p = predata->ILU_p;
835
836         precond pc_ilu;
837         pc_ilu.data = LU_p;
838         pc_ilu.fct  = fasp_precond_ilu;
839
840         fasp_solver_dcsr_pvfgmres(predata->S, &rs, deltaS, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
841
842     }
843
844 #else
845
846     fasp_dcsr_trans(predata->S,ptrA);
847     fasp_solver_umfpack(ptrA, &rs, deltaS, 0);
848     fasp_dcsr_free(ptrA);
849
850 #endif
851
852     //-------------------
854     //-------------------
855     fasp_blas_darray_axpy(colB, -1.0, deltaS->val, zs.val);
856
857     //-------------------
859     //-------------------
860     fasp_blas_dcsr_mxv(predata->Bt, deltaS->val, rv.val); // rv = B^T deltaS
861
862     for (i=0;i<colA;i++)
863     {
864         if (predata->diag_A->val[i] > SMALLREAL) rv.val[i]   = rv.val[i]/predata->
    diag_A->val[i]; // rv = D^{-1}rv
865     }
866
867     fasp_blas_darray_axpy (colA, -1.0, rv.val, zv.val); // zu = zu - rv
868
869     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
870     //-------------------
872     //-------------------
873     fasp_darray_cp(col, tempr, r);
874
875     // free
876     //fasp_dvec_free (&deltaS);
877
878 }
```

4.8.2.19 fasp_precond_ns_up_btri()

```
void fasp_precond_ns_up_btri (
             REAL * r,
             REAL * z,
             void * data )
```

back up r, setup z;

Solve Schur complement

Compute residule

Solve velocity

prepare AMG preconditioner

restore r

Definition at line 279 of file PreNavierStokes.c.

```
282 {
283     precond_ns_data *predata=(precond_ns_data *)data;
284     const int col = predata->col, colA = predata->colA, colB = predata->colB;
285     //const int maxit = predata->maxit;
286     //double *diagptr=predata->diag_S->val;
287
288     // local variables
289     double  *tempr = predata->w;
290
291     dvector rv; rv.row = colA; rv.val = r;
292     dvector zv; zv.row = colA; zv.val = z;
293     dvector rs; rs.row = colB; rs.val = r+colA;
294     dvector zs; zs.row = colB; zs.val = z+colA;
295
297     fasp_darray_cp(col, r, tempr);
298     fasp_darray_set(col, z, 0.0);
299
300     //-------------------------
302     //-------------------------
303     ITS_param *itparam_p = predata->ITS_param_p;
304
305 #if INEXACT
306
307     if (itparam_p->precond_type == 1) {
308         precond pc_s;
309         pc_s.data = predata->diag_S;
310         pc_s.fct  = fasp_precond_diag;
311         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
312     }
313     else if (itparam_p->precond_type == 2){
315         AMG_data *mgl_p = predata->mgl_data_p;
316         AMG_param *amgparam_p = predata->param_p;
317
318         precond_data pcdata_p;
319         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
320         pcdata_p.max_levels = mgl_p[0].num_levels;
321         pcdata_p.mgl_data = predata->mgl_data_p;
322         precond pc_p; pc_p.data = &pcdata_p;
323         pc_p.fct = fasp_precond_amg;
324
325         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
326     }
327     else if (itparam_p->precond_type == 4) {
328
329         ILU_data *LU_p = predata->ILU_p;
330
331         precond pc_ilu;
332         pc_ilu.data = LU_p;
```

```
333         pc_ilu.fct  = fasp_precond_ilu;
334
335         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
336
337     }
338
339 #else
340
341     dCSRmat tmpA;
342     dCSRmat *ptrA = &tmpA;
343     fasp_dcsr_trans(predata->S,ptrA);
344     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
345     fasp_dcsr_free(ptrA);
346
347 #endif
348
349     //--------------------
351     //--------------------
352     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
353
354     //--------------------
356     //--------------------
358     AMG_data *mgl_v = predata->mgl_data_v;
359     AMG_param *amgparam_v = predata->param_v;
360     ITS_param *itparam_v = predata->ITS_param_v;
361
362 #if INEXACT
363
364     precond_data pcdata_v;
365     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
366     pcdata_v.max_levels = mgl_v[0].num_levels;
367     pcdata_v.mgl_data = predata->mgl_data_v;
368     precond pc_v; pc_v.data = &pcdata_v;
369     pc_v.fct = fasp_precond_amg;
370
371     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
372
373 #else
374
375     //dCSRmat tmpA;
376     //dCSRmat *ptrA = &tmpA;
377     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
378     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
379     fasp_dcsr_free(ptrA);
380
381 #endif
382
384     fasp_darray_cp(col, tempr, r);
385 }
```

4.8.2.20 fasp_precond_ns_uzawa()

```
void fasp_precond_ns_uzawa (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute B zv - rs

Compute zs = omega∗(-1)∗(rs)

restore r

Definition at line 892 of file PreNavierStokes.c.

```
895 {
896
897     precond_ns_data *predata=(precond_ns_data *)data;
898     const int col = predata->col, colA = predata->colA, colB = predata->colB;
899
900     // local variables
901     double  *tempr = predata->w;
902
904     AMG_data *mgl_v = predata->mgl_data_v;
905     AMG_param *amgparam_v = predata->param_v;
906     ITS_param *itparam_v = predata->ITS_param_v;
907
908     dvector rv; rv.row = colA; rv.val = r;
909     dvector zv; zv.row = colA; zv.val = z;
910     dvector rs; rs.row = colB; rs.val = r+colA;
911     dvector zs; zs.row = colB; zs.val = z+colA;
912
914     fasp_darray_cp(col, r, tempr);
915     fasp_darray_set(col, z, 0.0);
916
917     //-------------------
919     //-------------------
920 #if INEXACT
921
922     precond_data pcdata_v;
923     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
924     pcdata_v.max_levels = mgl_v[0].num_levels;
925     pcdata_v.mgl_data = predata->mgl_data_v;
926     precond pc_v; pc_v.data = &pcdata_v;
927     pc_v.fct = fasp_precond_amg;
928
929     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
930
931     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
932 #else
933
934     dCSRmat tmpA;
935     dCSRmat *ptrA = &tmpA;
936     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
937     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
938     fasp_dcsr_free(ptrA);
939
940 #endif
941
942     //-------------------
944     //-------------------
945     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
946
947     //-------------------
949     //-------------------
950     REAL omega = -1.0;
951     fasp_blas_darray_axpy(colB, omega, rs.val,zs.val);
952
953     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
955     fasp_darray_cp(col, tempr, r);
956
957 }
```

4.8.2.21 fasp_precond_pnp_stokes_diag()

```
void fasp_precond_pnp_stokes_diag (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 36 of file PrePNPStokes.c.

```
39 {
40
41     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
42     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
43     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
44     dvector *tempr = &(precdata->r);
45
46     const INT N0 = A_pnp_csr->row;
47     const INT N1 = A_stokes_csr->row;
48     const INT N = N0 + N1;
49
50     // back up r, setup z;
51     fasp_darray_cp(N, r, tempr->val);
52     fasp_darray_set(N, z, 0.0);
53
54     // prepare
55 #if  WITH_UMFPACK
56     void **LU_diag = precdata->LU_diag;
57     dvector r0, r1, z0, z1;
58
59     r0.row = N0; z0.row = N0;
60     r1.row = N1; z1.row = N1;
61
62     r0.val = r; r1.val = &(r[N0]);
63     z0.val = z; z1.val = &(z[N0]);
64 #endif
65
66     // Preconditioning pnp block
67 #if  WITH_UMFPACK
68     /* use UMFPACK direct solver */
69     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
70 #endif
71
72     // Preconditioning A11 block
73 #if  WITH_UMFPACK
74     /* use UMFPACK direct solver */
75     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
76 #endif
77
78     // restore r
79     fasp_darray_cp(N, tempr->val, r);
80
81 }
```

4.8.2.22 fasp_precond_pnp_stokes_diag_inexact()

```
void fasp_precond_pnp_stokes_diag_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 219 of file PrePNPStokes.c.

```
222 {
223
224     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
225     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
226     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
227     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
228     dvector *tempr = &(precdata->r);
229
230     void **LU_diag = precdata->LU_diag;
231     precond_data_bsr *precdata_pnp = precdata->precdata_pnp;
232     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
233
234     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
235     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
236     const INT N = N0 + N1;
237
238     // back up r, setup z;
239     fasp_darray_cp(N, r, tempr->val);
240     fasp_darray_set(N, z, 0.0);
241
242     // prepare
243     dvector r0, r1, z0, z1;
244
245     r0.row = N0; z0.row = N0;
246     r1.row = N1; z1.row = N1;
247
248     r0.val = r; r1.val = &(r[N0]);
249     z0.val = z; z1.val = &(z[N0]);
250
251     // Preconditioning pnp block
252     //precond prec_pnp;
253
254     //prec_pnp.data = precdata_pnp;
255     //prec_pnp.fct = precdata->pnp_fct;
256
257     //prec_pnp.data = precdata->ILU_pnp;
258     //prec_pnp.fct = fasp_precond_dbsr_ilu;
259
260     //prec_pnp.data = precdata->ILU_pnp;
261     //prec_pnp.fct = fasp_precond_ilu;
262
263     //prec_pnp.data = precdata->diag_pnp;
264     //prec_pnp.fct = fasp_precond_dbsr_diag;
265
266     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
267     fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
268     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
269     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
270
271     // Preconditioning A11 block
272     precond prec_stokes;
273     prec_stokes.data = precdata_stokes;
274     prec_stokes.fct = precdata->stokes_fct;
275
276     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
277
278
279     // restore r
280     fasp_darray_cp(N, tempr->val, r);
281
282 }
```

4.8.2.23 fasp_precond_pnp_stokes_lower()

```
void fasp_precond_pnp_stokes_lower (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 95 of file PrePNPStokes.c.

```
98 {
99
100     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
101     dBLCmat *A = precdata->Abcsr;
102     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
103     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
104
105     dvector *tempr = &(precdata->r);
106
107     const INT N0 = A_pnp_csr->row;
108     const INT N1 = A_stokes_csr->row;
109     const INT N = N0 + N1;
110
111     // back up r, setup z;
112     fasp_darray_cp(N, r, tempr->val);
113     fasp_darray_set(N, z, 0.0);
114
115     // prepare
116     dvector r0, r1, z0, z1;
117
118     r0.row = N0; z0.row = N0;
119     r1.row = N1; z1.row = N1;
120
121     r0.val = r; r1.val = &(r[N0]);
122     z0.val = z; z1.val = &(z[N0]);
123
124     // Preconditioning pnp block
125 #if  WITH_UMFPACK
126     void **LU_diag = precdata->LU_diag;
127     /* use UMFPACK direct solver */
128     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
129 #endif
130
131     // r1 = r1 - A3*z0
132     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[2], z0.val, r1.val);
133
134     // Preconditioning stokes block
135 #if  WITH_UMFPACK
136     /* use UMFPACK direct solver */
137     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
138 #endif
139
140     // restore r
141     fasp_darray_cp(N, tempr->val, r);
142
143 }
```

4.8.2.24 fasp_precond_pnp_stokes_lower_inexact()

```
void fasp_precond_pnp_stokes_lower_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 296 of file PrePNPStokes.c.

```
299 {
300
301     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
302     dBLCmat *A = precdata->Abcsr;
303     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
304     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
305     //dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
306     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
307
308     dvector *tempr = &(precdata->r);
309
310     void **LU_diag = precdata->LU_diag;
311     precond_data_bsr *precdata_pnp= precdata->precdata_pnp;
312     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
313
314     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
315     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
316     const INT N = N0 + N1;
317
318     // back up r, setup z;
319     fasp_darray_cp(N, r, tempr->val);
320     fasp_darray_set(N, z, 0.0);
321
322     // prepare
323     dvector r0, r1, z0, z1;
324
325     r0.row = N0; z0.row = N0;
326     r1.row = N1; z1.row = N1;
327
328     r0.val = r; r1.val = &(r[N0]);
329     z0.val = z; z1.val = &(z[N0]);
330
331     // Preconditioning pnp block
332     //precond prec_pnp;
333
334     //prec_pnp.data = precdata_pnp;
335     //prec_pnp.fct = precdata->pnp_fct;
336
337     //prec_pnp.data = precdata->ILU_pnp;
338     //prec_pnp.fct = fasp_precond_dbsr_ilu;
339
340     //prec_pnp.data = precdata->ILU_pnp;
341     //prec_pnp.fct = fasp_precond_ilu;
```

```
342
343     //prec_pnp.data = precdata->diag_pnp;
344     //prec_pnp.fct = fasp_precond_dbsr_diag;
345
346     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
347     fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
348     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
349     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
350
351     // r1 = r1 - A3*z0
352     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[2], z0.val, r1.val);
353
354     // Preconditioning stokes block
355     precond prec_stokes;
356     prec_stokes.data = precdata_stokes;
357     prec_stokes.fct = precdata->stokes_fct;
358
359     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
360
361     // restore r
362     fasp_darray_cp(N, tempr->val, r);
363
364 }
```

4.8.2.25 fasp_precond_pnp_stokes_upper()

```
void fasp_precond_pnp_stokes_upper (
              REAL * r,
              REAL * z,
              void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 157 of file PrePNPStokes.c.

```
160 {
161
162     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
163     dBLCmat *A = precdata->Abcsr;
164     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
165     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
166
167     dvector *tempr = &(precdata->r);
168
169     const INT N0 = A_pnp_csr->row;
170     const INT N1 = A_pnp_csr->row;
```

```
171     const INT N = N0 + N1;
172
173     // back up r, setup z;
174     fasp_darray_cp(N, r, tempr->val);
175     fasp_darray_set(N, z, 0.0);
176
177     // prepare
178     dvector r0, r1, z0, z1;
179
180     r0.row = N0; z0.row = N0;
181     r1.row = N1; z1.row = N1;
182
183     r0.val = r; r1.val = &(r[N0]);
184     z0.val = z; z1.val = &(z[N0]);
185
186     // Preconditioning stokes block
187 #if  WITH_UMFPACK
188     void **LU_diag = precdata->LU_diag;
189     /* use UMFPACK direct solver */
190     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
191 #endif
192
193     // r1 = r1 - A5*z2
194     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[1], z1.val, r0.val);
195
196     // Preconditioning pnp block
197 #if  WITH_UMFPACK
198     /* use UMFPACK direct solver */
199     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
200 #endif
201
202     // restore r
203     fasp_darray_cp(N, tempr->val, r);
204
205 }
```

4.8.2.26 fasp_precond_pnp_stokes_upper_inexact()

```
void fasp_precond_pnp_stokes_upper_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 378 of file PrePNPStokes.c.

```
381 {
382
383     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
384     dBLCmat *A = precdata->Abcsr;
385     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
386     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
387     //dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
388     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
389
390     dvector *tempr = &(precdata->r);
391
392     void **LU_diag = precdata->LU_diag;
393     precond_data_bsr *precdata_pnp= precdata->precdata_pnp;
394     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
395
396     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
397     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
398     const INT N = N0 + N1;
399
400     // back up r, setup z;
401     fasp_darray_cp(N, r, tempr->val);
402     fasp_darray_set(N, z, 0.0);
403
404     // prepare
405     dvector r0, r1, z0, z1;
406
407     r0.row = N0; z0.row = N0;
408     r1.row = N1; z1.row = N1;
409
410     r0.val = r; r1.val = &(r[N0]);
411     z0.val = z; z1.val = &(z[N0]);
412
413     // Preconditioning stokes block
414     precond prec_stokes;
415     prec_stokes.data = precdata_stokes;
416     prec_stokes.fct = precdata->stokes_fct;
417
418     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
419
420     // r1 = r1 - A5*z2
421     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[1], z1.val, r0.val);
422
423     // Preconditioning pnp block
424     //precond prec_pnp;
425
426     //prec_pnp.data = precdata_pnp;
427     //prec_pnp.fct = precdata->pnp_fct;
428
429     //prec_pnp.data = precdata->ILU_pnp;
430     //prec_pnp.fct = fasp_precond_dbsr_ilu;
431
432     //prec_pnp.data = precdata->ILU_pnp;
433     //prec_pnp.fct = fasp_precond_ilu;
434
435     //prec_pnp.data = precdata->diag_pnp;
436     //prec_pnp.fct = fasp_precond_dbsr_diag;
437
438     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
439     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 1);
440     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
441     fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
442     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
443
444     // restore r
445     fasp_darray_cp(N, tempr->val, r);
446
447 }
```

4.8.2.27 fasp_solver_dblc_krylov_navier_stokes()

```
SHORT fasp_solver_dblc_krylov_navier_stokes (
              dBLCmat * Mat,
              dvector * b,
              dvector * x,
              itsolver_ns_param * itparam,
```

```
            AMG_ns_param * amgparam,
            ILU_param * iluparam,
            SWZ_param * schparam )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	pionter to AMG parameters for N-S
iluparam	pionter to ILU parameters
swzparam	pionter to Schwarz parameters

Returns

number of iterations

Author

Lu Wang

Date

03/02/2012

Modified by Xiaozhe Hu on 05/31/2016 Modified by Chensong Zhang on 03/15/2018

Definition at line 132 of file SolNavierStokes.c.

```
139 {
140     // parameters
141     const SHORT PrtLvl       = itparam->print_level;
142     const SHORT precond_type = itparam->precond_type;
143     const INT schwarz_mmsize = schparam->SWZ_mmsize;
144     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
145     const INT schwarz_type   = schparam->SWZ_type;
146
147 #if DEBUG_MODE > 0
148     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
149 #endif
150
151     // Navier-Stokes 2 by 2 matrix
152     dCSRmat *A  = Mat->blocks[0];
153     dCSRmat *Bt = Mat->blocks[1];
154     dCSRmat *B  = Mat->blocks[2];
155     dCSRmat *C  = Mat->blocks[3];
156
157     const INT n = A->row, m = B->row, nnzA = A->nnz;
158
159     // preconditioner data
160     dCSRmat *M = Mat->blocks[3];
161     dCSRmat S;
162     SWZ_data schwarz_data;
163     dvector diag_A;
164     dvector diag_S;
165     dCSRmat BABt;
166
167     // local variable
168     clock_t solver_start, solver_end, setup_start, setup_end;
169     REAL solver_duration, setup_duration;
170     SHORT status = FASP_SUCCESS;
```

```
171
172     //------ setup phase ------//
173     setup_start = clock();
174
175 #if DEBUG_MODE > 0
176     printf("### DEBUG: n = %d, m = %d, nnz = %d\n", n, m, nnzA);
177 #endif
178
179     //-------------------------//
180     // setup AMG for velocity  //
181     //-------------------------//
182     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
183     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA);
184
185     if (precond_type > 10) {
186
187         dCSRmat BtB;
188         fasp_blas_dcsr_mxm(Bt, B, &BtB);
189
190         REAL gamma = 10;
191         fasp_blas_dcsr_add (A, 1.0, &BtB, gamma, &mgl_v[0].A);
192
193         fasp_dcsr_free(&BtB);
194
195     }
196     else {
197         fasp_dcsr_cp(A,&mgl_v[0].A);
198     }
199
200     mgl_v[0].b = fasp_dvec_create(n);
201     mgl_v[0].x = fasp_dvec_create(n);
202
203     // setup AMG
204     switch (amgparam->param_v.AMG_type) {
205             case CLASSIC_AMG:
206             fasp_amg_setup_rs(mgl_v, &amgparam->param_v); break;
207             case SA_AMG:
208             fasp_amg_setup_sa(mgl_v, &amgparam->param_v); break;
209             case UA_AMG:
210             fasp_amg_setup_ua(mgl_v, &amgparam->param_v); break;
211             default:
212             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
213             exit(ERROR_INPUT_PAR);
214     }
215
216     // get diagonal of A
217     fasp_dcsr_getdiag(n, &mgl_v[0].A, &diag_A);
218
219     //---------------------------//
220     // setup Schur complement S  //
221     //---------------------------//
222
223     if ( precond_type == 8  || precond_type == 9 ||
224         precond_type == 18 || precond_type == 19 ) {
225
226         fasp_blas_dcsr_mxm(B, Bt, &S);
227
228         // change the sign of the BB^T
229         fasp_blas_dcsr_axm(&S, -1.0);
230
231         // make it non-singular
232         INT i,k,j,ibegin,iend;
233
234         for (i=0;i<S.row;++i) {
235             ibegin=S.IA[i]; iend=S.IA[i+1];
236             for (k=ibegin;k<iend;++k) {
237                 j=S.JA[k];
238                 if ( j==i ) {
239                     S.val[k] = S.val[k] + 1e-8; break;
240                 } // end if
241             } // end for k
242         } // end for i
243
244     }
245     else if ( precond_type == 10 || precond_type == 20 ) {
246
247         fasp_blas_dcsr_mxm(B, Bt, &S);
248         fasp_blas_dcsr_rap(B, A, Bt, &BABt);
249
250         // change the sign of the BB^T
251         fasp_blas_dcsr_axm(&S, -1.0);
252
253         // make it non-singular
254         INT i,k,j,ibegin,iend;
255
256         for (i=0;i<S.row;++i) {
257             ibegin=S.IA[i]; iend=S.IA[i+1];
```

```
258             for (k=ibegin;k<iend;++k) {
259                 j=S.JA[k];
260                 if ( j==i ) {
261                     S.val[k] = S.val[k] + 1e-8; break;
262                 } // end if
263             } // end for k
264         } // end for i
265 
266     }
267     else {
268         get_schur_diagA(B,Bt,A,C,&S);
269     }
270 
271     dvector res_p = fasp_dvec_create(m);
272     dvector sol_p = fasp_dvec_create(m);
273 
274     AMG_data *mgl_p;
275     ILU_data LU_p;
276 
277     if ( itparam->precond_type_p == 1 ) {
278         fasp_dcsr_getdiag(0,&S,&diag_S);
279     }
280     else if ( itparam->precond_type_p == 2 ) {
281         // Setup AMG for Schur Complement
282         dCSRmat *As = &S;
283         const INT nnzS = As->nnz;
284         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
285         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
286         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
287         // setup AMG
288         switch (amgparam->param_p.AMG_type) {
289                 case CLASSIC_AMG:
290                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p); break;
291                 case SA_AMG:
292                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p); break;
293                 case UA_AMG:
294                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p); break;
295                 default:
296                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
297                        amgparam->param_p.AMG_type);
298                 exit(ERROR_INPUT_PAR);
299         }
300     }
301     else if ( itparam->precond_type_p == 4 ) {
302         // setup ILU for Schur Complement
303         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
304         fasp_mem_iludata_check(&LU_p);
305     }
306 
307     //-----------------------------------//
308     // Setup itsolver parameter for subblocks
309     //-----------------------------------//
310     ITS_param ITS_param_v;
311     fasp_param_solver_init(&ITS_param_v);
312     ITS_param_v.print_level = itparam->print_level_v;
313     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
314     ITS_param_v.restart = itparam->pre_restart_v;
315     ITS_param_v.tol = itparam->pre_tol_v;
316     ITS_param_v.maxit = itparam->pre_maxit_v;
317     ITS_param_v.precond_type = itparam->precond_type_v;
318 
319     ITS_param ITS_param_p;
320     fasp_param_solver_init(&ITS_param_p);
321     ITS_param_p.print_level = itparam->print_level_p;
322     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
323     ITS_param_p.restart = itparam->pre_restart_p;
324     ITS_param_p.tol = itparam->pre_tol_p;
325     ITS_param_p.maxit = itparam->pre_maxit_p;
326     ITS_param_p.precond_type = itparam->precond_type_p;
327 
328     //-------------------------//
329     // setup preconditioner
330     //-------------------------//
331     precond prec;
332     precond_ns_data precdata;
333     prec.data = &precdata;
334 
335     precdata.colA = n;
336     precdata.colB = m;
337     precdata.col  = n+m;
338     precdata.M    = M;
339     precdata.B    = B;
340     precdata.Bt   = Bt;
341     precdata.C    = C;
342     precdata.BABt = &BABt;
343 
344     precdata.param_v     = &amgparam->param_v;
```

```
345     precdata.param_p     = &amgparam->param_p;
346     precdata.ITS_param_v = &ITS_param_v;
347     precdata.ITS_param_p = &ITS_param_p;
348     precdata.mgl_data_v  = mgl_v;
349     precdata.mgl_data_p  = mgl_p;
350     precdata.ILU_p       = &LU_p;
351
352     precdata.max_levels     = mgl_v[0].num_levels;
353     precdata.print_level    = amgparam->param_v.print_level;
354     precdata.maxit          = amgparam->param_v.maxit;
355     precdata.amg_tol        = amgparam->param_v.tol;
356     precdata.cycle_type     = amgparam->param_v.cycle_type;
357     precdata.smoother       = amgparam->param_v.smoother;
358     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
359     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
360     precdata.relaxation     = amgparam->param_v.relaxation;
361     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
362
363     precdata.diag_A = &diag_A;
364     precdata.S = &S;
365     precdata.diag_S = &diag_S;
366     precdata.rp = &res_p;
367     precdata.sp = &sol_p;
368     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
369
370     switch (precond_type) {
371             case 1:
372             prec.fct = fasp_precond_ns_bdiag;
373             break;
374             case 2:
375             prec.fct = fasp_precond_ns_low_btri;
376             break;
377             case 3:
378             prec.fct = fasp_precond_ns_up_btri;
379             break;
380             case 4:
381             prec.fct = fasp_precond_ns_blu;
382             break;
383             case 5:
384             prec.fct = fasp_precond_ns_simple;
385             break;
386             case 6:
387             prec.fct = fasp_precond_ns_simpler;
388             break;
389             case 7:
390             prec.fct = fasp_precond_ns_uzawa;
391             break;
392             case 8:
393             prec.fct = fasp_precond_ns_projection;
394             break;
395             case 9:
396             prec.fct = fasp_precond_ns_DGS;
397             break;
398             case 10:
399             prec.fct = fasp_precond_ns_LSCDGS;
400             break;
401             case 11:
402             prec.fct = fasp_precond_ns_bdiag;
403             break;
404             case 12:
405             prec.fct = fasp_precond_ns_low_btri;
406             break;
407             case 13:
408             prec.fct = fasp_precond_ns_up_btri;
409             break;
410             case 14:
411             prec.fct = fasp_precond_ns_blu;
412             break;
413             case 15:
414             prec.fct = fasp_precond_ns_simple;
415             break;
416             case 16:
417             prec.fct = fasp_precond_ns_simpler;
418             break;
419             case 17:
420             prec.fct = fasp_precond_ns_uzawa;
421             break;
422             case 18:
423             prec.fct = fasp_precond_ns_projection;
424             break;
425             case 19:
426             prec.fct = fasp_precond_ns_DGS;
427             break;
428             case 20:
429             prec.fct = fasp_precond_ns_LSCDGS;
430             break;
431             default:
```

```
432             printf("### ERROR: Unknown preconditioner type!\n");
433             exit(ERROR_SOLVER_PRECTYPE);
434     }
435
436     setup_end = clock();
437
438     if (PrtLvl>0) {
439         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
440         printf("Setup costs %f.\n", setup_duration);
441     }
442
443     //------ solve phase ------//
444     solver_start=clock();
445     //status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
446     status=fasp_ns_solver_itsolver(Mat,b,x,NULL,itparam);
447     solver_end=clock();
448
449     if (PrtLvl>0) {
450         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
451         printf(COLOR_RESET);
452         printf("Solver costs %f seconds.\n", solver_duration);
453         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
454     }
455
456     //FINISHED:
457     // clean up memory
458     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
459     if (itparam->precond_type_p == 1) fasp_dvec_free(&diag_S);
460     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
461
462     fasp_mem_free(precdata.w);
463     fasp_dvec_free(&res_p);
464     fasp_dvec_free(&sol_p);
465     fasp_dcsr_free(&S);
466     if (precond_type == 10 || precond_type == 20) fasp_dcsr_free(&BABt);
467     fasp_dvec_free(&diag_A);
468
469     return status;
470 }
```

4.8.2.28 fasp_solver_dblc_krylov_navier_stokes_pmass()

```
SHORT fasp_solver_dblc_krylov_navier_stokes_pmass (
              dBLCmat * Mat,
              dvector * b,
              dvector * x,
              itsolver_ns_param * itparam,
              AMG_ns_param * amgparam,
              ILU_param * iluparam,
              SWZ_param * schparam,
              dCSRmat * Mp )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	AMG parameters for NS
iluparam	ILU parameters
schparam	Schwarz parameters
precdata	pionter to preconditioner data for ns
Mp	pointer to dCSRmat of the pressure mass matrix

Returns

number of iterations

Author

Xiaozhe Hu

Date

017/07/2014

Note

In general, this is for purely Stokes problem, NS problem with div-div stablization – Xiaozhe

Definition at line 499 of file SolNavierStokes.c.

```
507 {
508     // parameters
509     const SHORT PrtLvl = itparam->print_level;
510     const SHORT precond_type = itparam->precond_type;
511     const INT schwarz_mmsize = schparam->SWZ_mmsize;
512     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
513     const INT schwarz_type   = schparam->SWZ_type;
514
515     // Navier-Stokes 4 by 4 matrix
516     dCSRmat *A  = Mat->blocks[0];
517     dCSRmat *Bt = Mat->blocks[1];
518     dCSRmat *B  = Mat->blocks[2];
519     dCSRmat *C  = Mat->blocks[3];
520     const INT n = A->row, m = B->row, nnzA = A->nnz;
521
522     // preconditioner data
523     dCSRmat *M = Mat->blocks[3];
524     dCSRmat S,P;
525     SWZ_data schwarz_data;
526     dvector diag_S;
527
528     // local variable
529     clock_t solver_start, solver_end, setup_start, setup_end;
530     REAL    solver_duration, setup_duration;
531     SHORT   status = FASP_SUCCESS;
532
533 #if DEBUG_MODE > 0
534     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
535 #endif
536
537     //------ setup phase ------//
538     setup_start = clock();
539
540     //-----------------------//
541     // setup AMG for velocity
542     //-----------------------//
543     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
544     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA); fasp_dcsr_cp(A,&mgl_v[0].A);
545     mgl_v[0].b=fasp_dvec_create(n); mgl_v[0].x=fasp_dvec_create(n);
546
547     // setup AMG
548     switch (amgparam->param_v.AMG_type) {
549             case CLASSIC_AMG:
550             fasp_amg_setup_rs(mgl_v, &amgparam->param_v);
551             break;
552             case SA_AMG:
553             fasp_amg_setup_sa(mgl_v, &amgparam->param_v);
554             break;
555             case UA_AMG:
556             fasp_amg_setup_ua(mgl_v, &amgparam->param_v);
557             break;
558             default:
559             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
560             exit(ERROR_INPUT_PAR);
561     }
562
563     //-------------------------//
```

```
564     // setup Schur complement S using pressure mass
565     //-------------------------//
566
567     fasp_dcsr_alloc(Mp->row, Mp->col, Mp->nnz, &S);
568     fasp_dcsr_cp(Mp, &S);
569
570     dvector res_p = fasp_dvec_create(m);
571     dvector sol_p = fasp_dvec_create(m);
572
573     AMG_data *mgl_p;
574     ILU_data LU_p;
575
576     if ( itparam->precond_type_p == 1 ) {
577         fasp_dcsr_getdiag(0,&S,&diag_S);
578     }
579     else if ( itparam->precond_type_p == 2 ) {
580         // setup AMG for Schur Complement
581         dCSRmat *As = &S;
582         const INT nnzS = As->nnz;
583         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
584         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
585         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
586         // setup AMG
587         switch ( amgparam->param_p.AMG_type ) {
588                 case CLASSIC_AMG:
589                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p);
590                 break;
591                 case SA_AMG:
592                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p);
593                 break;
594                 case UA_AMG:
595                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p);
596                 break;
597                 default:
598                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
599                       amgparam->param_p.AMG_type);
600                 exit(ERROR_INPUT_PAR);
601         }
602     }
603     else if ( itparam->precond_type_p == 4 ) {
604         // setup ILU for Schur Complement
605         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
606         fasp_mem_iludata_check(&LU_p);
607     }
608
609     //---------------------------------------//
610     // Setup itsolver parameter for subblocks
611     //---------------------------------------//
612     ITS_param ITS_param_v;
613     fasp_param_solver_init(&ITS_param_v);
614     ITS_param_v.print_level = itparam->print_level_v;
615     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
616     ITS_param_v.restart = itparam->pre_restart_v;
617     ITS_param_v.tol = itparam->pre_tol_v;
618     ITS_param_v.maxit = itparam->pre_maxit_v;
619     ITS_param_v.precond_type = itparam->precond_type_v;
620
621     ITS_param ITS_param_p;
622     fasp_param_solver_init(&ITS_param_p);
623     ITS_param_p.print_level = itparam->print_level_p;
624     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
625     ITS_param_p.restart = itparam->pre_restart_p;
626     ITS_param_p.tol = itparam->pre_tol_p;
627     ITS_param_p.maxit = itparam->pre_maxit_p;
628     ITS_param_p.precond_type = itparam->precond_type_p;
629
630     //-------------------------//
631     // setup preconditioner
632     //-------------------------//
633     precond prec;
634     precond_ns_data precdata;
635     prec.data = &precdata;
636
637     precdata.colA = n;
638     precdata.colB = m;
639     precdata.col  = n+m;
640     precdata.M    = M;
641     precdata.B    = B;
642     precdata.Bt   = Bt;
643     precdata.C    = C;
644
645     precdata.param_v        = &amgparam->param_v;
646     precdata.param_p        = &amgparam->param_p;
647     precdata.ITS_param_v    = &ITS_param_v;
648     precdata.ITS_param_p    = &ITS_param_p;
649     precdata.mgl_data_v     = mgl_v;
650     precdata.mgl_data_p     = mgl_p;
```

```
651     precdata.ILU_p          = &LU_p;
652
653     precdata.max_levels     = mgl_v[0].num_levels;
654     precdata.print_level    = amgparam->param_v.print_level;
655     precdata.maxit          = amgparam->param_v.maxit;
656     precdata.amg_tol        = amgparam->param_v.tol;
657     precdata.cycle_type     = amgparam->param_v.cycle_type;
658     precdata.smoother       = amgparam->param_v.smoother;
659     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
660     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
661     precdata.relaxation     = amgparam->param_v.relaxation;
662     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
663
664
665     precdata.S = &S;
666     precdata.diag_S = &diag_S;
667     precdata.rp = &res_p;
668     precdata.sp = &sol_p;
669
670     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
671
672     switch (precond_type) {
673             case 1:
674             prec.fct = fasp_precond_ns_bdiag; break;
675             case 2:
676             prec.fct = fasp_precond_ns_low_btri; break;
677             case 3:
678             prec.fct = fasp_precond_ns_up_btri; break;
679             case 4:
680             prec.fct = fasp_precond_ns_blu; break;
681             default:
682             printf("### ERROR: Unknown preconditioner type!\n");
683             exit(ERROR_SOLVER_PRECTYPE);
684     }
685
686     setup_end = clock();
687
688     if (PrtLvl>0) {
689         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
690         printf("Setup costs %f.\n", setup_duration);
691     }
692
693     //------ solver phase ------//
694     solver_start=clock();
695     status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
696     solver_end=clock();
697
698     if (PrtLvl>0) {
699         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
700         printf(COLOR_RESET);
701         printf("Solver costs %f seconds.\n", solver_duration);
702         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
703     }
704
705     // clean up memory
706     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
707     if (itparam->precond_type_p == 1) fasp_dvec_free(&diag_S);
708     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
709
710     fasp_mem_free(precdata.w);
711     fasp_dvec_free(&res_p);
712     fasp_dvec_free(&sol_p);
713     fasp_dcsr_free(&S);
714
715     return status;
716 }
```

4.8.2.29 fasp_solver_dblc_krylov_navier_stokes_schur_pmass()

```
SHORT fasp_solver_dblc_krylov_navier_stokes_schur_pmass (
              dBLCmat * Mat,
              dvector * b,
              dvector * x,
              itsolver_ns_param * itparam,
              AMG_ns_param * amgparam,
```

```
            ILU_param * iluparam,
            SWZ_param * schparam,
            dCSRmat * Mp )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	AMG parameters for NS
iluparam	ILU parameters
schparam	Schwarz parameters
precdata	pionter to preconditioner data for ns
Mp	pointer to dCSRmat of the pressure mass matrix

Returns

number of iterations

Author

Xiaozhe Hu

Date

017/07/2014

Note

In general, this is for NS problems without div-div stablization and pressure stablization (pressure block is zero), moreover, pressure mass matrix is provided.

Definition at line 747 of file SolNavierStokes.c.

```
755 {
756 #if DEBUG_MODE > 0
757     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
758 #endif
759
760     // parameters
761     const SHORT PrtLvl = itparam->print_level;
762     const SHORT precond_type = itparam->precond_type;
763     const INT schwarz_mmsize = schparam->SWZ_mmsize;
764     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
765     const INT schwarz_type   = schparam->SWZ_type;
766
767     // Navier-Stokes 4 by 4 matrix
768     dCSRmat *A  = Mat->blocks[0];
769     dCSRmat *Bt = Mat->blocks[1];
770     dCSRmat *B  = Mat->blocks[2];
771     dCSRmat *C  = Mat->blocks[3];
772     const INT n = A->row, m = B->row, nnzA = A->nnz;
773
774     // preconditioner data
775     dCSRmat *M = Mat->blocks[3];
776     dCSRmat S,P;
777     SWZ_data schwarz_data;
```

```
778     dvector diag_S;
779
780     // local variable
781     clock_t solver_start, solver_end, setup_start, setup_end;
782     REAL solver_duration, setup_duration;
783     SHORT status=FASP_SUCCESS;
784
785     //------ setup phase ------//
786     setup_start = clock();
787
788     //-----------------------//
789     // setup AMG for velocity
790     //-----------------------//
791
792     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
793
794     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA); fasp_dcsr_cp(A,&mgl_v[0].A);
795     mgl_v[0].b=fasp_dvec_create(n); mgl_v[0].x=fasp_dvec_create(n);
796
797     // setup AMG
798     switch (amgparam->param_v.AMG_type) {
799             case CLASSIC_AMG:
800             fasp_amg_setup_rs(mgl_v, &amgparam->param_v);
801             break;
802             case SA_AMG:
803             fasp_amg_setup_sa(mgl_v, &amgparam->param_v);
804             break;
805             case UA_AMG:
806             fasp_amg_setup_ua(mgl_v, &amgparam->param_v);
807             break;
808             default:
809             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
810             exit(ERROR_INPUT_PAR);
811     }
812
813     //-------------------------//
814     // setup Schur complement S using pressure mass
815     //-------------------------//
816
817     get_schur_pmass(B, Bt, &mgl_v[0].A, Mp, 1e5, &S);
818     // TODO: 1e5 is a parameter can be tuned, not sure how to tune now -- Xiaozhe
819
820     dvector res_p = fasp_dvec_create(m);
821     dvector sol_p = fasp_dvec_create(m);
822
823     AMG_data *mgl_p;
824     ILU_data LU_p;
825
826     if (itparam->precond_type_p == 1){
827         fasp_dcsr_getdiag(0,&S,&diag_S);
828     }
829     else if (itparam->precond_type_p == 2) {
830         // Setup AMG for Schur Complement
831         dCSRmat *As = &S;
832         const INT nnzS = As->nnz;
833         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
834         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
835         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
836         // setup AMG
837         switch (amgparam->param_p.AMG_type) {
838                 case CLASSIC_AMG:
839                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p);
840                 break;
841                 case SA_AMG:
842                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p);
843                 break;
844                 case UA_AMG:
845                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p);
846                 break;
847                 default:
848                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
849                         amgparam->param_p.AMG_type);
850                 exit(ERROR_INPUT_PAR);
851         }
852     }
853     else if (itparam->precond_type_p == 4) {
854         // setup ILU for Schur Complement
855         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
856         fasp_mem_iludata_check(&LU_p);
857     }
858
859     //---------------------------------------//
860     // Setup itsolver parameter for subblocks
861     //---------------------------------------//
862     ITS_param ITS_param_v;
863     fasp_param_solver_init(&ITS_param_v);
864     ITS_param_v.print_level = itparam->print_level_v;
```

```
865     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
866     ITS_param_v.restart = itparam->pre_restart_v;
867     ITS_param_v.tol = itparam->pre_tol_v;
868     ITS_param_v.maxit = itparam->pre_maxit_v;
869     ITS_param_v.precond_type = itparam->precond_type_v;
870
871     ITS_param ITS_param_p;
872     fasp_param_solver_init(&ITS_param_p);
873     ITS_param_p.print_level = itparam->print_level_p;
874     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
875     ITS_param_p.restart = itparam->pre_restart_p;
876     ITS_param_p.tol = itparam->pre_tol_p;
877     ITS_param_p.maxit = itparam->pre_maxit_p;
878     ITS_param_p.precond_type = itparam->precond_type_p;
879
880     //-------------------------//
881     // setup preconditioner
882     //-------------------------//
883     precond prec;
884     precond_ns_data precdata;
885     prec.data = &precdata;
886
887     precdata.colA = n;
888     precdata.colB = m;
889     precdata.col  = n+m;
890     precdata.M    = M;
891     precdata.B    = B;
892     precdata.Bt   = Bt;
893     precdata.C    = C;
894
895     precdata.param_v        = &amgparam->param_v;
896     precdata.param_p        = &amgparam->param_p;
897     precdata.ITS_param_v    = &ITS_param_v;
898     precdata.ITS_param_p    = &ITS_param_p;
899     precdata.mgl_data_v     = mgl_v;
900     precdata.mgl_data_p     = mgl_p;
901     precdata.ILU_p          = &LU_p;
902
903     precdata.max_levels     = mgl_v[0].num_levels;
904     precdata.print_level    = amgparam->param_v.print_level;
905     precdata.maxit          = amgparam->param_v.maxit;
906     precdata.amg_tol        = amgparam->param_v.tol;
907     precdata.cycle_type     = amgparam->param_v.cycle_type;
908     precdata.smoother       = amgparam->param_v.smoother;
909     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
910     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
911     precdata.relaxation     = amgparam->param_v.relaxation;
912     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
913
914     precdata.S = &S;
915     precdata.diag_S = &diag_S;
916     precdata.rp = &res_p;
917     precdata.sp = &sol_p;
918
919     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
920
921     switch (precond_type) {
922             case 1:
923             prec.fct = fasp_precond_ns_bdiag;
924             break;
925             case 2:
926             prec.fct = fasp_precond_ns_low_btri;
927             break;
928             case 3:
929             prec.fct = fasp_precond_ns_up_btri;
930             break;
931             case 4:
932             prec.fct = fasp_precond_ns_blu;
933             break;
934             default:
935             printf("### ERROR: Unknown preconditioner type!\n");
936             exit(ERROR_SOLVER_PRECTYPE);
937     }
938
939     setup_end = clock();
940
941     if (PrtLvl>0) {
942         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
943         printf("Setup costs %f.\n", setup_duration);
944     }
945
946     //------ solver phase ------//
947     solver_start=clock();
948     status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
949     solver_end=clock();
950
951     if (PrtLvl>0) {
```

```
952         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
953         printf(COLOR_RESET);
954         printf("Solver costs %f seconds.\n", solver_duration);
955         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
956     }
957
958     //FINISHED:
959     // clean up memory
960     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
961     if (itparam->precond_type_p == 1){fasp_dvec_free(&diag_S);}
962     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
963
964     fasp_mem_free(precdata.w);
965     fasp_dvec_free(&res_p);
966     fasp_dvec_free(&sol_p);
967     fasp_dcsr_free (&S);
968
969     return status;
970 }
```

4.8.2.30 fasp_solver_dblc_krylov_pnp_stokes()

```
INT fasp_solver_dblc_krylov_pnp_stokes (
              dBLCmat * A,
              dvector * b,
              dvector * x,
              ITS_param * itparam,
              ITS_param * itparam_pnp,
              AMG_param * amgparam_pnp,
              itsolver_ns_param * itparam_stokes,
              AMG_ns_param * amgparam_stokes,
              const int num_velocity,
              const int num_pressure )
```

Solve Ax = b by standard Krylov methods.

Parameters

A	Pointer to the coeff matrix in dBLCmat format
b	Pointer to the right hand side in dvector format
x	Pointer to the approx solution in dvector format
itparam	Pointer to parameters for iterative solvers
amgparam	Pointer to parameters for AMG solvers

Returns

Iteration number if converges; ERROR otherwise.

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 52 of file SolPNPStokes.c.

```
62 {
63     const SHORT prtlvl = itparam->print_level;
64     const SHORT precond_type = itparam->precond_type;
65
66     INT status = FASP_SUCCESS;
67     REAL setup_start, setup_end, setup_duration;
68     REAL solver_start, solver_end, solver_duration;
69
70     INT m, n, nnz, i, k;
71
72     // local variables
73     dCSRmat A_pnp_csr;
74     dBSRmat A_pnp_bsr;
75
76     dCSRmat A_stokes_csr;
77     dBLCmat A_stokes_bcsr;
78     dCSRmat S;
79     dCSRmat BABt;
80
81     // data for pnp
82     AMG_data_bsr *mgl_pnp=fasp_amg_data_bsr_create(amgparam_pnp->max_levels);
83     ILU_param iluparam_pnp;
84     iluparam_pnp.print_level = amgparam_pnp->print_level;
85     iluparam_pnp.ILU_lfil    = amgparam_pnp->ILU_lfil;
86     iluparam_pnp.ILU_droptol = amgparam_pnp->ILU_droptol;
87     iluparam_pnp.ILU_relax   = amgparam_pnp->ILU_relax;
88     iluparam_pnp.ILU_type    = amgparam_pnp->ILU_type;
89     ILU_data ILU_pnp;
90
91     // data for stokes
92     AMG_data *mgl_v = fasp_amg_data_create(amgparam_stokes->param_v.max_levels);
93     AMG_data *mgl_p;
94     dvector   res_p = fasp_dvec_create(num_pressure);
95     dvector   sol_p = fasp_dvec_create(num_pressure);
96
97 #if WITH_UMFPACK
98     void **LU_diag = (void **)fasp_mem_calloc(2, sizeof(void *));
99 #endif
100
101 #if DEBUG_MODE > 0
102     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
103 #endif
104
105     /* setup preconditioner */
106     fasp_gettime(&setup_start);
107
108     /* diagonal blocks are solved exactly */
109     if ( precond_type > 20 && precond_type < 30 ) {
110 #if WITH_UMFPACK
111         // Need to sort the diagonal blocks for UMFPACK format
112         // pnp block
113         A_pnp_csr = fasp_dcsr_create(A->blocks[0]->row, A->blocks[0]->col, A->blocks[0]->nnz);
114         fasp_dcsr_transz(A->blocks[0], NULL, &A_pnp_csr);
115
116         printf("Factorization for pnp diagonal block: \n");
117         LU_diag[0] = fasp_umfpack_factorize(&A_pnp_csr, prtlvl);
118
119         // stokes block
120         A_stokes_csr = fasp_dcsr_create(A->blocks[3]->row, A->blocks[3]->col, A->blocks[3]->nnz);
121         fasp_dcsr_transz(A->blocks[3], NULL, &A_stokes_csr);
122
123         printf("Factorization for stokes diagonal block: \n");
124         LU_diag[1] = fasp_umfpack_factorize(&A_stokes_csr, prtlvl);
125 #endif
126     }
127
128     /* diagonal blocks are solved inexactly */
129     else if ( precond_type > 30 && precond_type < 40 ) {
130
131         // pnp block
132         {
133             A_pnp_bsr = fasp_format_dcsr_dbsr(A->blocks[0], 3);
134
135             // AMG for pnp
136             /*
137              // initialize A, b, x for mgl_pnp[0]
138              mgl_pnp[0].A = fasp_dbsr_create(A_pnp_bsr.ROW, A_pnp_bsr.COL, A_pnp_bsr.NNZ, A_pnp_bsr.nb,
      A_pnp_bsr.storage_manner);
```

```
139             mgl_pnp[0].b = fasp_dvec_create(mgl_pnp[0].A.ROW*mgl_pnp[0].A.nb);
140             mgl_pnp[0].x = fasp_dvec_create(mgl_pnp[0].A.COL*mgl_pnp[0].A.nb);
141
142             fasp_dbsr_cp(&A_pnp_bsr, &(mgl_pnp[0].A));
143
144             switch (amgparam_pnp->AMG_type) {
145
146             case SA_AMG: // Smoothed Aggregation AMG
147             status = fasp_amg_setup_sa_bsr(mgl_pnp, amgparam_pnp); break;
148
149             default:
150             status = fasp_amg_setup_ua_bsr(mgl_pnp, amgparam_pnp); break;
151
152             }
153
154             if (status < 0) goto FINISHED;
155             */
156
157            // diagonal preconditioner for pnp
158            /*
159             // diag of the pnp matrix
160             fasp_dvec_alloc(A_pnp_bsr.ROW*A_pnp_bsr.nb*A_pnp_bsr.nb, &diag_pnp);
161             for (i = 0; i < A_pnp_bsr.ROW; ++i) {
162             for (k = A_pnp_bsr.IA[i]; k < A_pnp_bsr.IA[i+1]; ++k) {
163             if (A_pnp_bsr.JA[k] == i)
164             memcpy(diag_pnp.val+i*A_pnp_bsr.nb*A_pnp_bsr.nb, A_pnp_bsr.val+k*A_pnp_bsr.nb*A_pnp_bsr.nb,
    A_pnp_bsr.nb*A_pnp_bsr.nb*sizeof(REAL));
165             }
166             }
167
168             for (i=0; i<A_pnp_bsr.ROW; ++i){
169             fasp_blas_smat_inv(&(diag_pnp.val[i*A_pnp_bsr.nb*A_pnp_bsr.nb]), A_pnp_bsr.nb);
170             }
171             */
172
173            // BSR ILU for pnp
174            /*
175             // ILU setup
176             if ( (status = fasp_ilu_dbsr_setup(&A_pnp_bsr, &ILU_pnp, &iluparam_pnp)) < 0 ) goto FINISHED;
177
178             // check iludata
179             if ( (status = fasp_mem_iludata_check(&ILU_pnp)) < 0 ) goto FINISHED;
180             */
181
182            // CSR ILU for pnp
183            /*
184             // ILU setup for whole matrix
185             if ( (status = fasp_ilu_dcsr_setup(A->blocks[0],&ILU_pnp,&iluparam_pnp)) < 0 ) goto FINISHED;
186
187             // check iludata
188             if ( (status = fasp_mem_iludata_check(&ILU_pnp)) < 0 ) goto FINISHED;
189             */
190
191        }
192
193        // stokes block
194        {
195            A_stokes_bcsr.brow = 2;
196            A_stokes_bcsr.bcol = 2;
197            A_stokes_bcsr.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
198            for (i=0; i<4 ;i++) {
199                A_stokes_bcsr.blocks[i] = (dCSRmat *)fasp_mem_calloc(1, sizeof(dCSRmat));
200            }
201
202            ivector velocity_idx;
203            ivector pressure_idx;
204            fasp_ivec_alloc(num_velocity, &velocity_idx);
205            fasp_ivec_alloc(num_pressure, &pressure_idx);
206            for (i=0; i<num_velocity; i++) velocity_idx.val[i] = i;
207            for (i=0; i<num_pressure; i++) pressure_idx.val[i] = num_velocity + i;
208
209            fasp_dcsr_getblk(A->blocks[3], velocity_idx.val, velocity_idx.val, velocity_idx.row,
    velocity_idx.row, A_stokes_bcsr.blocks[0]);
210            fasp_dcsr_getblk(A->blocks[3], velocity_idx.val, pressure_idx.val, velocity_idx.row,
    pressure_idx.row, A_stokes_bcsr.blocks[1]);
211            fasp_dcsr_getblk(A->blocks[3], pressure_idx.val, velocity_idx.val, pressure_idx.row,
    velocity_idx.row, A_stokes_bcsr.blocks[2]);
212            fasp_dcsr_getblk(A->blocks[3], pressure_idx.val, pressure_idx.val, pressure_idx.row,
    pressure_idx.row, A_stokes_bcsr.blocks[3]);
213
214            fasp_ivec_free(&velocity_idx);
215            fasp_ivec_free(&pressure_idx);
216
217
218            // AMG for velocity
219            mgl_v[0].A=fasp_dcsr_create(A_stokes_bcsr.blocks[0]->row,A_stokes_bcsr.blocks[0]->col,
    A_stokes_bcsr.blocks[0]->nnz);
```

```
220             fasp_dcsr_cp(A_stokes_bcsr.blocks[0], &mgl_v[0].A);
221             mgl_v[0].b=fasp_dvec_create(A_stokes_bcsr.blocks[0]->row); mgl_v[0].x=fasp_dvec_create(
    A_stokes_bcsr.blocks[0]->col);
222
223             switch (amgparam_stokes->param_v.AMG_type) {
224                 case CLASSIC_AMG:
225                     fasp_amg_setup_rs(mgl_v, &amgparam_stokes->param_v);
226                     break;
227                 case SA_AMG:
228                     fasp_amg_setup_sa(mgl_v, &amgparam_stokes->param_v);
229                     break;
230                 case UA_AMG:
231                     fasp_amg_setup_ua(mgl_v, &amgparam_stokes->param_v);
232                     break;
233                 default:
234                     printf("Error: Wrong AMG type %d!\n",amgparam_stokes->param_v.AMG_type);
235                     exit(ERROR_INPUT_PAR);
236             }
237
238
239             //-------------------------//
240             // setup Schur complement S
241             //-------------------------//
242             fasp_blas_dcsr_mxm(A_stokes_bcsr.blocks[2], A_stokes_bcsr.blocks[1], &S);
243             fasp_blas_dcsr_rap(A_stokes_bcsr.blocks[2], A_stokes_bcsr.blocks[0], A_stokes_bcsr.blocks[1], &
    BABt);
244
245             // change the sign of the BB^T
246             fasp_blas_dcsr_axm(&S, -1.0);
247
248             // make it non-singular
249             INT k,j,ibegin,iend;
250
251             for (i=0;i<S.row;++i) {
252                 ibegin=S.IA[i]; iend=S.IA[i+1];
253                 for (k=ibegin;k<iend;++k) {
254                     j=S.JA[k];
255                     if ((j-i)==0) {
256                         S.val[k] = S.val[k] + 1e-8; break;
257                     } // end if
258                 } // end for k
259             } // end for i
260
261             dCSRmat *As = &S;
262             const int nnzS = As->nnz;
263             mgl_p=fasp_amg_data_create(amgparam_stokes->param_p.max_levels);
264             mgl_p[0].A=fasp_dcsr_create(num_pressure,num_pressure,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
265             mgl_p[0].b=fasp_dvec_create(num_pressure); mgl_p[0].x=fasp_dvec_create(num_pressure);
266             // setup AMG
267             switch (amgparam_stokes->param_p.AMG_type) {
268                 case CLASSIC_AMG:
269                     fasp_amg_setup_rs(mgl_p, &amgparam_stokes->param_p);
270                     break;
271                 case SA_AMG:
272                     fasp_amg_setup_sa(mgl_p, &amgparam_stokes->param_p);
273                     break;
274                 case UA_AMG:
275                     fasp_amg_setup_ua(mgl_p, &amgparam_stokes->param_p);
276                     break;
277                 default:
278                     printf("Error: Wrong AMG type %d for Schur Complement!\n",amgparam_stokes->param_p.
    AMG_type);
279                     exit(ERROR_INPUT_PAR);
280             }
281
282         }
283
284     }
285
286     else {
287         fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
288     }
289
290     // generate data for preconditioners
291     // data for pnp
292     precond_data_bsr precdata_pnp;
293     precdata_pnp.print_level = amgparam_pnp->print_level;
294     precdata_pnp.maxit = amgparam_pnp->maxit;
295     precdata_pnp.tol = amgparam_pnp->tol;
296     precdata_pnp.cycle_type = amgparam_pnp->cycle_type;
297     precdata_pnp.smoother = amgparam_pnp->smoother;
298     precdata_pnp.presmooth_iter = amgparam_pnp->presmooth_iter;
299     precdata_pnp.postsmooth_iter = amgparam_pnp->postsmooth_iter;
300     precdata_pnp.coarsening_type = amgparam_pnp->coarsening_type;
301     precdata_pnp.relaxation = amgparam_pnp->relaxation;
302     precdata_pnp.coarse_scaling = amgparam_pnp->coarse_scaling;
303     precdata_pnp.amli_degree = amgparam_pnp->amli_degree;
```

```
304     precdata_pnp.amli_coef = amgparam_pnp->amli_coef;
305     precdata_pnp.tentative_smooth = amgparam_pnp->tentative_smooth;
306     precdata_pnp.max_levels = mgl_pnp[0].num_levels;
307     precdata_pnp.mgl_data = mgl_pnp;
308     precdata_pnp.A = &A_pnp_bsr;
309
310     // data for stokes
311     // Setup itsolver parameters
312     ITS_param ITS_param_v;
313     fasp_param_solver_init(&ITS_param_v);
314     ITS_param_v.print_level = itparam_stokes->print_level_v;
315     ITS_param_v.itsolver_type = itparam_stokes->itsolver_type_v;
316     ITS_param_v.restart = itparam_stokes->pre_restart_v;
317     ITS_param_v.tol = itparam_stokes->pre_tol_v;
318     ITS_param_v.maxit = itparam_stokes->pre_maxit_v;
319     ITS_param_v.precond_type = itparam_stokes->precond_type_v;
320
321     ITS_param ITS_param_p;
322     fasp_param_solver_init(&ITS_param_p);
323     ITS_param_p.print_level = itparam_stokes->print_level_p;
324     ITS_param_p.itsolver_type = itparam_stokes->itsolver_type_p;
325     ITS_param_p.restart = itparam_stokes->pre_restart_p;
326     ITS_param_p.tol = itparam_stokes->pre_tol_p;
327     ITS_param_p.maxit = itparam_stokes->pre_maxit_p;
328     ITS_param_p.precond_type = itparam_stokes->precond_type_p;
329
330     // data for stokes
331     precond_ns_data  precdata_stokes;
332     if ( precond_type > 30 && precond_type < 40 ) {
333         precdata_stokes.colA = A_stokes_bcsr.blocks[0]->row;
334         precdata_stokes.colB = A_stokes_bcsr.blocks[2]->row;
335         precdata_stokes.col  = A_stokes_bcsr.blocks[0]->row + A_stokes_bcsr.blocks[2]->row;
336         precdata_stokes.B  = A_stokes_bcsr.blocks[2];
337         precdata_stokes.Bt = A_stokes_bcsr.blocks[1];
338         precdata_stokes.C = A_stokes_bcsr.blocks[3];
339         precdata_stokes.BABt  = &BABt;
340     }
341
342     precdata_stokes.param_v         = &amgparam_stokes->param_v;
343     precdata_stokes.param_p         = &amgparam_stokes->param_p;
344     precdata_stokes.ITS_param_v     = &ITS_param_v;
345     precdata_stokes.ITS_param_p     = &ITS_param_p;
346     precdata_stokes.mgl_data_v      = mgl_v;
347     precdata_stokes.mgl_data_p      = mgl_p;
348
349     precdata_stokes.max_levels      = mgl_v[0].num_levels;
350     precdata_stokes.print_level     = amgparam_stokes->param_v.print_level;
351     precdata_stokes.maxit           = amgparam_stokes->param_v.maxit;
352     precdata_stokes.amg_tol         = amgparam_stokes->param_v.tol;
353     precdata_stokes.cycle_type      = amgparam_stokes->param_v.cycle_type;
354     precdata_stokes.smoother        = amgparam_stokes->param_v.smoother;
355     precdata_stokes.presmooth_iter  = amgparam_stokes->param_v.presmooth_iter;
356     precdata_stokes.postsmooth_iter = amgparam_stokes->param_v.postsmooth_iter;
357     precdata_stokes.relaxation      = amgparam_stokes->param_v.relaxation;
358     precdata_stokes.coarse_scaling  = amgparam_stokes->param_v.coarse_scaling;
359
360     precdata_stokes.S  = &S;
361     precdata_stokes.rp = &res_p;
362     precdata_stokes.sp = &sol_p;
363
364     precdata_stokes.w = (double *)fasp_mem_calloc(precdata_stokes.col,sizeof(double));
365
366     // data for overall
367     precond_pnp_stokes_data precdata;
368     precdata.Abcsr = A;
369
370 #if WITH_UMFPACK
371     // LU if exact solve
372     precdata.LU_diag = LU_diag;
373 #endif
374
375     // pnp part
376     precdata.A_pnp_csr = &A_pnp_csr;
377     precdata.A_pnp_bsr = &A_pnp_bsr;
378     precdata.precdata_pnp = &precdata_pnp;
379     precdata.pnp_fct = fasp_precond_dbsr_amg;
380     precdata.ILU_pnp = &ILU_pnp;
381
382     // stokes part
383     precdata.A_stokes_csr = &A_stokes_csr;
384     precdata.A_stokes_bcsr = &A_stokes_bcsr;
385     precdata.precdata_stokes = &precdata_stokes;
386     precdata.stokes_fct = fasp_precond_ns_LSCDGS;
387
388     precdata.r = fasp_dvec_create(b->row);
389
390     precond prec; prec.data = &precdata;
```

```
391
392     switch (precond_type)
393     {
394         case 21:
395             prec.fct = fasp_precond_pnp_stokes_diag;
396             break;
397
398         case 22:
399             prec.fct = fasp_precond_pnp_stokes_lower;
400             break;
401
402         case 23:
403             prec.fct = fasp_precond_pnp_stokes_upper;
404             break;
405
406         case 31:
407             prec.fct = fasp_precond_pnp_stokes_diag_inexact;
408             break;
409
410         case 32:
411             prec.fct = fasp_precond_pnp_stokes_lower_inexact;
412             break;
413
414         case 33:
415             prec.fct = fasp_precond_pnp_stokes_upper_inexact;
416             break;
417
418         default:
419             fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
420             break;
421     }
422
423     if ( prtlvl >= PRINT_MIN ) {
424         fasp_gettime(&setup_end);
425         setup_duration = setup_end - setup_start;
426         fasp_cputime("Setup totally", setup_duration);
427     }
428
429
430     // solver part
431     fasp_gettime(&solver_start);
432
433     status=fasp_solver_dblc_itsolver(A,b,x, &prec,itparam);
434
435     fasp_gettime(&solver_end);
436
437     solver_duration = solver_end - solver_start;
438
439     if ( prtlvl >= PRINT_MIN )
440         fasp_cputime("Krylov method totally", solver_duration);
441
442 FINISHED:
443
444     // clean
445     /* diagonal blocks are solved exactly */
446     if ( precond_type > 20 && precond_type < 30 ) {
447 #if WITH_UMFPACK
448         for (i=0; i<2; i++) fasp_umfpack_free_numeric(LU_diag[i]);
449
450         fasp_dcsr_free(&A_pnp_csr);
451         fasp_dcsr_free(&A_stokes_csr);
452
453         fasp_dvec_free(&precdata.r);
454 #endif
455     }
456     /* diagonal blocks are solved by AMG */
457     else if (precond_type > 30 && precond_type < 40) {
458 #if WITH_UMFPACK
459         for (i=0; i<2; i++) fasp_umfpack_free_numeric(LU_diag[i]);
460 #endif
461         fasp_dbsr_free(&A_pnp_bsr);
462         fasp_amg_data_bsr_free(mgl_pnp);
463         //if (&ILU_pnp) fasp_ilu_data_free(&ILU_pnp);
464
465         fasp_dblc_free(&A_stokes_bcsr);
466         fasp_dcsr_free(&S);
467         fasp_dcsr_free(&BABt);
468         fasp_amg_data_free(mgl_v, &amgparam_stokes->param_v);
469         fasp_amg_data_free(mgl_p, &amgparam_stokes->param_p);
470         fasp_dvec_free(&res_p);
471         fasp_dvec_free(&sol_p);
472         fasp_mem_free(precdata_stokes.w);
473
474         fasp_dvec_free(&precdata.r);
475
476     }
477
```

```
478     else {
479         fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
480     }
481
482 #if DEBUG_MODE > 0
483     printf("### DEBUG: %s ..... [Finish]\n", __FUNCTION__);
484 #endif
485
486     return status;
487 }
```

4.9 functs.inl File Reference

Basis functions and problem information.

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

Functions

- double u (double x, double y)
  true solution u

4.9.1 Detailed Description

Basis functions and problem information.

Copyright (C) 2011–2018 by the FASP team. All rights reserved.

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

4.9.2 Function Documentation

4.9.2.1 u()

```
double u (
            double x,
            double y )
```

true solution u

true solution p

true solution v

Parameters

x	the x-axis value of the point
y	the y-axis value of the point

Returns

function value

Author

Lu Wang

Date

11/30/2011

Definition at line 86 of file functs.inl.

```
87 {
88     return -cos(x)*sin(y);
89 }
```

4.10 PreNavierStokes.c File Reference

Preconditioners for (Navier-)Stokes problems.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- void fasp_precond_ns_bdiag (REAL *r, REAL *z, void *data)

  block diagonal preconditioning for ns equation
- void fasp_precond_ns_low_btri (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_up_btri (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_blu (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_simple (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_simpler (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_uzawa (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_projection (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_DGS (REAL *r, REAL *z, void *data)
- void fasp_precond_ns_LSCDGS (REAL *r, REAL *z, void *data)

4.10.1 Detailed Description

Preconditioners for (Navier-)Stokes problems.

Note

This file contains Level-4 (Pre) functions.

// TODO: Fix Doxygen. –Chensong

4.10.2 Function Documentation

4.10.2.1 fasp_precond_ns_bdiag()

```
void fasp_precond_ns_bdiag (
             REAL * r,
             REAL * z,
             void * data )
```

block diagonal preconditioning for ns equation

Parameters

∗r	pointer to residual
∗z	pointer to preconditioned residual
∗data	pointer to precondition data

Author

Xiaozhe Hu, Lu Wang

Date

10/20/2013

Note

modified by Lu Wang on 02/12/2014
Xiaozhe Hu modified on 02/21/2014
: modified by Xiaozhe Hu on May. 27, 2014

setup z;

Solve velocity

prepare AMG preconditioner

Solve Schur complement

Definition at line 41 of file PreNavierStokes.c.

```
44 {
45     precond_ns_data *predata=(precond_ns_data *)data;
46 
47     const INT col = predata->col, colA = predata->colA, colB = predata->colB;
48 
49     dvector rv; rv.row = colA; rv.val = r;
50     dvector zv; zv.row = colA; zv.val = z;
51     dvector rs; rs.row = colB; rs.val = r+colA;
52     dvector zs; zs.row = colB; zs.val = z+colA;
53 
55     fasp_darray_set(col, z, 0.0);
56 
57     //--------------------
59     //--------------------
61     AMG_data *mgl_v = predata->mgl_data_v;
62     AMG_param *amgparam_v = predata->param_v;
63     ITS_param *itparam_v = predata->ITS_param_v;
64 
65 #if INEXACT
66 
67     precond_data pcdata_v;
68     fasp_param_amg_to_prec(&pcdata_v, amgparam_v);
69     pcdata_v.max_levels = mgl_v[0].num_levels;
70     pcdata_v.mgl_data = predata->mgl_data_v;
71     precond pc_v; pc_v.data = &pcdata_v;
72     pc_v.fct = fasp_precond_amg;
73 
74     if(itparam_v->print_level > 0) printf(COLOR_RESET "\n");
75     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
76 
77 #else
78 
79     dCSRmat tmpA;
80     dCSRmat *ptrA = &tmpA;
81     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
82     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
83     fasp_dcsr_free(ptrA);
84 
85 #endif
86 
87 
88     //-------------------------
90     //-------------------------
91     ITS_param *itparam_p = predata->ITS_param_p;
92 
93 #if INEXACT
94 
95     if (itparam_p->precond_type == 1) {
96         precond pc_s;
97         pc_s.data = predata->diag_S;
98         pc_s.fct  = fasp_precond_diag;
99         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
100     }
101     else if (itparam_p->precond_type == 2){
103         AMG_data *mgl_p = predata->mgl_data_p;
104         AMG_param *amgparam_p = predata->param_p;
105 
106         precond_data pcdata_p;
107         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
108         pcdata_p.max_levels = mgl_p[0].num_levels;
109         pcdata_p.mgl_data = predata->mgl_data_p;
110         precond pc_p; pc_p.data = &pcdata_p;
111         pc_p.fct = fasp_precond_amg;
112 
113         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
114     }
115     else if (itparam_p->precond_type == 4) {
116 
117         ILU_data *LU_p = predata->ILU_p;
118 
119         precond pc_ilu;
120         pc_ilu.data = LU_p;
121         pc_ilu.fct  = fasp_precond_ilu;
122 
123         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
124 
125     }
126 
127 #else
128 
129     fasp_dcsr_trans(predata->S,ptrA);
130     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
131     fasp_dcsr_free(ptrA);
```

```
132
133 #endif
134
135     if(itparam_v->print_level > 0)
136         printf(COLOR_GREEN "\n");
137
138 }
```

4.10.2.2 fasp_precond_ns_blu()

```
void fasp_precond_ns_blu (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute residule

Solve velocity

restore r

Definition at line 399 of file PreNavierStokes.c.

```
402 {
403
404     precond_ns_data *predata=(precond_ns_data *)data;
405     const int col = predata->col, colA = predata->colA, colB = predata->colB;
406
407     // local variables
408     double  *tempr = predata->w;
409
411     AMG_data *mgl_v = predata->mgl_data_v;
412     AMG_param *amgparam_v = predata->param_v;
413     ITS_param *itparam_v = predata->ITS_param_v;
414
415     dvector rv; rv.row = colA; rv.val = r;
416     dvector zv; zv.row = colA; zv.val = z;
417     dvector rs; rs.row = colB; rs.val = r+colA;
418     dvector zs; zs.row = colB; zs.val = z+colA;
419
421     fasp_darray_cp(col, r, tempr);
422     fasp_darray_set(col, z, 0.0);
423
424     //--------------------
426     //--------------------
427 #if INEXACT
428
429     precond_data pcdata_v;
430     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
431     pcdata_v.max_levels = mgl_v[0].num_levels;
432     pcdata_v.mgl_data = predata->mgl_data_v;
433     precond pc_v; pc_v.data = &pcdata_v;
434     pc_v.fct = fasp_precond_amg;
435
436     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
437
438     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
```

```
    restart, 1, itparam_v->print_level);
439 #else
440
441     dCSRmat tmpA;
442     dCSRmat *ptrA = &tmpA;
443     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
444     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
445     fasp_dcsr_free(ptrA);
446
447 #endif
448
449     //--------------------
451     //--------------------
452     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
453
454     //-------------------------
456     //-------------------------
457     ITS_param *itparam_p = predata->ITS_param_p;
458
459 #if INEXACT
460
461     if (itparam_p->precond_type == 1) {
462         precond pc_s;
463         pc_s.data = predata->diag_S;
464         pc_s.fct  = fasp_precond_diag;
465         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
466     }
467     else if (itparam_p->precond_type == 2){
469         AMG_data *mgl_p = predata->mgl_data_p;
470         AMG_param *amgparam_p = predata->param_p;
471
472         precond_data pcdata_p;
473         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
474         pcdata_p.max_levels = mgl_p[0].num_levels;
475         pcdata_p.mgl_data = predata->mgl_data_p;
476         precond pc_p; pc_p.data = &pcdata_p;
477         pc_p.fct = fasp_precond_amg;
478
479         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
480     }
481     else if (itparam_p->precond_type == 4) {
482
483         ILU_data *LU_p = predata->ILU_p;
484
485         precond pc_ilu;
486         pc_ilu.data = LU_p;
487         pc_ilu.fct  = fasp_precond_ilu;
488
489         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
490
491     }
492
493 #else
494
495     fasp_dcsr_trans(predata->S,ptrA);
496     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
497     fasp_dcsr_free(ptrA);
498
499 #endif
500
501     //--------------------
503     //--------------------
504     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
505
506     //--------------------
508     //--------------------
509 #if INEXACT
510
511     fasp_darray_set(colA, zv.val, 0.0);
512
513     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
514
515     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
516 #else
517
518     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
519     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
520     fasp_dcsr_free(ptrA);
521
522 #endif
523
524     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
526     fasp_darray_cp(col, tempr, r);
```

```
527
528 }
```

4.10.2.3 fasp_precond_ns_DGS()

```
void fasp_precond_ns_DGS (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement -BB^T

Compute zv = zv - B^T * sp

Compute zs = zs + BB^T * sp

restore r

Definition at line 1104 of file PreNavierStokes.c.

```
1107 {
1108     precond_ns_data *predata=(precond_ns_data *)data;
1109     const int col = predata->col, colA = predata->colA, colB = predata->colB;
1110
1111     // local variables
1112     double  *tempr = predata->w;
1113
1115     AMG_data *mgl_v = predata->mgl_data_v;
1116     AMG_param *amgparam_v = predata->param_v;
1117     ITS_param *itparam_v = predata->ITS_param_v;
1118
1119     dvector rv; rv.row = colA; rv.val = r;
1120     dvector zv; zv.row = colA; zv.val = z;
1121     dvector rs; rs.row = colB; rs.val = r+colA;
1122     dvector zs; zs.row = colB; zs.val = z+colA;
1123
1125     fasp_darray_cp(col, r, tempr);
1126     fasp_darray_set(col, z, 0.0);
1127
1128     //--------------------
1130     //--------------------
1131 #if INEXACT
1132
1133     precond_data pcdata_v;
1134     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1135     pcdata_v.max_levels = mgl_v[0].num_levels;
1136     pcdata_v.mgl_data = predata->mgl_data_v;
1137     precond pc_v; pc_v.data = &pcdata_v;
1138     pc_v.fct = fasp_precond_amg;
1139
1140     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1141
1142     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
1143 #else
1144
1145     dCSRmat tmpA;
1146     dCSRmat *ptrA = &tmpA;
1147     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
```

```
1148     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1149     fasp_dcsr_free(ptrA);
1150
1151 #endif
1152
1153     //--------------------
1155     //--------------------
1156     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1157
1158     //--------------------------
1160     //--------------------------
1161     ITS_param *itparam_p = predata->ITS_param_p;
1162
1163 #if INEXACT
1164
1165     if (itparam_p->precond_type == 1) {
1166         precond pc_s;
1167         pc_s.data = predata->diag_S;
1168         pc_s.fct  = fasp_precond_diag;
1169         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_s, itparam_p->tol,itparam_p->maxit,
     itparam_p->restart, 1, itparam_p->print_level);
1170     }
1171     else if (itparam_p->precond_type == 2){
1173         AMG_data *mgl_p = predata->mgl_data_p;
1174         AMG_param *amgparam_p = predata->param_p;
1175
1176         precond_data pcdata_p;
1177         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1178         pcdata_p.max_levels = mgl_p[0].num_levels;
1179         pcdata_p.mgl_data = predata->mgl_data_p;
1180         precond pc_p; pc_p.data = &pcdata_p;
1181         pc_p.fct = fasp_precond_amg;
1182
1183         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, predata->sp, &pc_p, itparam_p->tol,itparam_p->maxit,
     itparam_p->restart, 1, itparam_p->print_level);
1184     }
1185     else if (itparam_p->precond_type == 4) {
1186
1187         ILU_data *LU_p = predata->ILU_p;
1188
1189         precond pc_ilu;
1190         pc_ilu.data = LU_p;
1191         pc_ilu.fct  = fasp_precond_ilu;
1192
1193         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_ilu, itparam_p->tol,itparam_p->maxit
     , itparam_p->restart, 1, itparam_p->print_level);
1194
1195     }
1196
1197 #else
1198
1199     //dCSRmat tmpA;
1200     //dCSRmat *ptrA = &tmpA;
1201     fasp_dcsr_trans(predata->S,ptrA);
1202     fasp_solver_umfpack(ptrA, &rs, predata->sp, 0);
1203     fasp_dcsr_free(ptrA);
1204
1205 #endif
1206
1207     //--------------------
1209     //--------------------
1210     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, predata->sp->val, zv.val);
1211
1212     //--------------------
1214     //--------------------
1215     fasp_blas_dcsr_aAxpy(1.0, predata->S, predata->sp->val, zs.val);
1216
1217
1218     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1220     fasp_darray_cp(col, tempr, r);
1221 }
```

4.10.2.4 fasp_precond_ns_low_btri()

```
void fasp_precond_ns_low_btri (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

restore r

Definition at line 155 of file PreNavierStokes.c.

```
158 {
159     precond_ns_data *predata=(precond_ns_data *)data;
160     const int col = predata->col, colA = predata->colA, colB = predata->colB;
161
162     // local variables
163     double  *tempr = predata->w;
164
166     AMG_data *mgl_v = predata->mgl_data_v;
167     AMG_param *amgparam_v = predata->param_v;
168     ITS_param *itparam_v = predata->ITS_param_v;
169
170     dvector rv; rv.row = colA; rv.val = r;
171     dvector zv; zv.row = colA; zv.val = z;
172     dvector rs; rs.row = colB; rs.val = r+colA;
173     dvector zs; zs.row = colB; zs.val = z+colA;
174
176     fasp_darray_cp(col, r, tempr);
177     fasp_darray_set(col, z, 0.0);
178
179     //--------------------
181     //--------------------
182 #if INEXACT
183
184     precond_data pcdata_v;
185     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
186     pcdata_v.max_levels = mgl_v[0].num_levels;
187     pcdata_v.mgl_data = predata->mgl_data_v;
188     precond pc_v; pc_v.data = &pcdata_v;
189     pc_v.fct = fasp_precond_amg;
190
191     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
192
193     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
194 #else
195
196     dCSRmat tmpA;
197     dCSRmat *ptrA = &tmpA;
198     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
199     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
200     fasp_dcsr_free(ptrA);
201
202 #endif
203
204     //--------------------
206     //--------------------
207     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
208
209     //-------------------------
211     //-------------------------
212     ITS_param *itparam_p = predata->ITS_param_p;
213
214 #if INEXACT
215
216     if (itparam_p->precond_type == 1) {
217         precond pc_s;
218         pc_s.data = predata->diag_S;
219         pc_s.fct  = fasp_precond_diag;
220         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
221     }
222     else if (itparam_p->precond_type == 2){
224         AMG_data *mgl_p = predata->mgl_data_p;
225         AMG_param *amgparam_p = predata->param_p;
226
227         precond_data pcdata_p;
```

```
228         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
229         pcdata_p.max_levels = mgl_p[0].num_levels;
230         pcdata_p.mgl_data = predata->mgl_data_p;
231         precond pc_p; pc_p.data = &pcdata_p;
232         pc_p.fct = fasp_precond_amg;
233 
234         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
235     }
236     else if (itparam_p->precond_type == 4) {
237 
238         ILU_data *LU_p = predata->ILU_p;
239 
240         precond pc_ilu;
241         pc_ilu.data = LU_p;
242         pc_ilu.fct  = fasp_precond_ilu;
243 
244         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
245 
246     }
247 
248 #else
249 
250     //dCSRmat tmpA;
251     //dCSRmat *ptrA = &tmpA;
252     fasp_dcsr_trans(predata->S,ptrA);
253     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
254     fasp_dcsr_free(ptrA);
255 
256 #endif
257 
258     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
260     fasp_darray_cp(col, tempr, r);
261 }
```

4.10.2.5 fasp_precond_ns_LSCDGS()

```
void fasp_precond_ns_LSCDGS (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute zv = zv + B^T * sp

Compute rs = BAB^t * sp

Solve Schur complement

Compute zs = -zs

restore r

Definition at line 1235 of file PreNavierStokes.c.

```
1238 {
1239     precond_ns_data *predata=(precond_ns_data *)data;
1240     const int col = predata->col, colA = predata->colA, colB = predata->colB;
1241 
1242     // local variables
1243     double  *tempr = predata->w;
1244 
1246     AMG_data *mgl_v = predata->mgl_data_v;
1247     AMG_param *amgparam_v = predata->param_v;
1248     ITS_param *itparam_v = predata->ITS_param_v;
1249 
1250     dvector rv; rv.row = colA; rv.val = r;
1251     dvector zv; zv.row = colA; zv.val = z;
1252     dvector rs; rs.row = colB; rs.val = r+colA;
1253     dvector zs; zs.row = colB; zs.val = z+colA;
1254 
1256     fasp_darray_cp(col, r, tempr);
1257     fasp_darray_set(col, z, 0.0);
1258 
1259     //--------------------
1261     //--------------------
1262 #if INEXACT
1263 
1264     precond_data pcdata_v;
1265     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1266     pcdata_v.max_levels = mgl_v[0].num_levels;
1267     pcdata_v.mgl_data = predata->mgl_data_v;
1268     precond pc_v; pc_v.data = &pcdata_v;
1269     pc_v.fct = fasp_precond_amg;
1270 
1271     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1272 
1273     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
1274 #else
1275 
1276     dCSRmat tmpA;
1277     dCSRmat *ptrA = &tmpA;
1278     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
1279     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1280     fasp_dcsr_free(ptrA);
1281 
1282 #endif
1283 
1284     //--------------------
1286     //--------------------
1287     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1288 
1289     //--------------------------
1291     //--------------------------
1292     ITS_param *itparam_p = predata->ITS_param_p;
1293 
1294 #if INEXACT
1295 
1296     if (itparam_p->precond_type == 1) {
1297         precond pc_s;
1298         pc_s.data = predata->diag_S;
1299         pc_s.fct  = fasp_precond_diag;
1300         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_s, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
1301     }
1302     else if (itparam_p->precond_type == 2){
1304         AMG_data *mgl_p = predata->mgl_data_p;
1305         AMG_param *amgparam_p = predata->param_p;
1306 
1307         precond_data pcdata_p;
1308         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1309         pcdata_p.max_levels = mgl_p[0].num_levels;
1310         pcdata_p.mgl_data = predata->mgl_data_p;
1311         precond pc_p; pc_p.data = &pcdata_p;
1312         pc_p.fct = fasp_precond_amg;
1313 
1314         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, predata->sp, &pc_p, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
1315     }
1316     else if (itparam_p->precond_type == 4) {
1317 
1318         ILU_data *LU_p = predata->ILU_p;
1319 
1320         precond pc_ilu;
1321         pc_ilu.data = LU_p;
1322         pc_ilu.fct  = fasp_precond_ilu;
1323 
1324         fasp_solver_dcsr_pvfgmres(predata->S, &rs, predata->sp, &pc_ilu, itparam_p->tol,itparam_p->maxit
    , itparam_p->restart, 1, itparam_p->print_level);
1325 
1326     }
```

```
1327
1328     /*
1329      fasp_dcoo_write("Ap.dat",  predata->S);
1330      fasp_dvec_write("rp.dat", &rs);
1331      getchar();
1332      */
1333
1334 #else
1335
1336     //dCSRmat tmpA;
1337     //dCSRmat *ptrA = &tmpA;
1338     fasp_dcsr_trans(predata->S,ptrA);
1339     fasp_solver_umfpack(ptrA, &rs, predata->sp, 0);
1340     fasp_dcsr_free(ptrA);
1341
1342 #endif
1343
1344     // change the sign of the solution
1345     fasp_blas_darray_ax(predata->sp->row, -1.0, predata->sp->val);
1346
1347     //--------------------
1349     //--------------------
1350     fasp_blas_dcsr_aAxpy(1.0, predata->Bt, predata->sp->val, zv.val);
1351
1352     //--------------------
1354     //--------------------
1355     fasp_blas_dcsr_mxv(predata->BABt, predata->sp->val, rs.val);
1356
1357     //--------------------------
1359     //--------------------------
1360
1361 #if INEXACT
1362
1363     if (itparam_p->precond_type == 1) {
1364         precond pc_s;
1365         pc_s.data = predata->diag_S;
1366         pc_s.fct  = fasp_precond_diag;
1367         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
1368     }
1369     else if (itparam_p->precond_type == 2){
1371         AMG_data *mgl_p = predata->mgl_data_p;
1372         AMG_param *amgparam_p = predata->param_p;
1373
1374         precond_data pcdata_p;
1375         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1376         pcdata_p.max_levels = mgl_p[0].num_levels;
1377         pcdata_p.mgl_data = predata->mgl_data_p;
1378         precond pc_p; pc_p.data = &pcdata_p;
1379         pc_p.fct = fasp_precond_amg;
1380
1381         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
1382     }
1383     else if (itparam_p->precond_type == 4) {
1384
1385         ILU_data *LU_p = predata->ILU_p;
1386
1387         precond pc_ilu;
1388         pc_ilu.data = LU_p;
1389         pc_ilu.fct  = fasp_precond_ilu;
1390
1391         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
1392
1393     }
1394
1395 #else
1396
1397     //dCSRmat tmpA;
1398     //dCSRmat *ptrA = &tmpA;
1399     fasp_dcsr_trans(predata->S,ptrA);
1400     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
1401     fasp_dcsr_free(ptrA);
1402
1403 #endif
1404
1405     //--------------------
1407     //--------------------
1408     //fasp_blas_darray_ax(colB,-1.0,zs.val);
1409
1410
1411     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1413     fasp_darray_cp(col, tempr, r);
1414 }
```

4.10.2.6 fasp_precond_ns_projection()

```
void fasp_precond_ns_projection (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement -B*B^T

Compute zv = zv - B^T * zs

Compute zs = sp

restore r

Definition at line 971 of file PreNavierStokes.c.

```
974 {
975     precond_ns_data *predata=(precond_ns_data *)data;
976     const int col = predata->col, colA = predata->colA, colB = predata->colB;
977
978     // local variables
979     double  *tempr = predata->w;
980
982     AMG_data *mgl_v = predata->mgl_data_v;
983     AMG_param *amgparam_v = predata->param_v;
984     ITS_param *itparam_v = predata->ITS_param_v;
985
986     dvector rv; rv.row = colA; rv.val = r;
987     dvector zv; zv.row = colA; zv.val = z;
988     dvector rs; rs.row = colB; rs.val = r+colA;
989     dvector zs; zs.row = colB; zs.val = z+colA;
990
992     fasp_darray_cp(col, r, tempr);
993     fasp_darray_set(col, z, 0.0);
994
995     //-------------------
997     //-------------------
998 #if INEXACT
999
1000     precond_data pcdata_v;
1001     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
1002     pcdata_v.max_levels = mgl_v[0].num_levels;
1003     pcdata_v.mgl_data = predata->mgl_data_v;
1004     precond pc_v; pc_v.data = &pcdata_v;
1005     pc_v.fct = fasp_precond_amg;
1006
1007     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
1008
1009     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
1010 #else
1011
1012     dCSRmat tmpA;
1013     dCSRmat *ptrA = &tmpA;
1014     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
1015     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
1016     fasp_dcsr_free(ptrA);
1017
1018 #endif
1019
```

```
1020     //--------------------
1022     //--------------------
1023     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
1024     fasp_darray_cp(colB, rs.val, predata->sp->val);
1025
1026     //--------------------------
1028     //--------------------------
1029     ITS_param *itparam_p = predata->ITS_param_p;
1030
1031 #if INEXACT
1032
1033     if (itparam_p->precond_type == 1) {
1034         precond pc_s;
1035         pc_s.data = predata->diag_S;
1036         pc_s.fct  = fasp_precond_diag;
1037         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
1038     }
1039     else if (itparam_p->precond_type == 2){
1041         AMG_data *mgl_p = predata->mgl_data_p;
1042         AMG_param *amgparam_p = predata->param_p;
1043
1044         precond_data pcdata_p;
1045         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
1046         pcdata_p.max_levels = mgl_p[0].num_levels;
1047         pcdata_p.mgl_data = predata->mgl_data_p;
1048         precond pc_p; pc_p.data = &pcdata_p;
1049         pc_p.fct = fasp_precond_amg;
1050
1051         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
1052     }
1053     else if (itparam_p->precond_type == 4) {
1054
1055         ILU_data *LU_p = predata->ILU_p;
1056
1057         precond pc_ilu;
1058         pc_ilu.data = LU_p;
1059         pc_ilu.fct  = fasp_precond_ilu;
1060
1061         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
1062
1063     }
1064
1065 #else
1066
1067     //dCSRmat tmpA;
1068     //dCSRmat *ptrA = &tmpA;
1069     fasp_dcsr_trans(predata->S,ptrA);
1070     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
1071     fasp_dcsr_free(ptrA);
1072
1073 #endif
1074
1075     //--------------------
1077     //--------------------
1078     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, zv.val);
1079
1080     //--------------------
1082     //--------------------
1083     fasp_darray_cp(colB, predata->sp->val, zs.val);
1084
1085
1086     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
1088     fasp_darray_cp(col, tempr, r);
1089 }
```

4.10.2.7 fasp_precond_ns_simple()

```
void fasp_precond_ns_simple (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute residule

Solve Schur complement

Compute zu = zu - D^{-1}B^T zs

restore r

Definition at line 542 of file PreNavierStokes.c.

```
545 {
546
547     precond_ns_data *predata=(precond_ns_data *)data;
548     const int col = predata->col, colA = predata->colA, colB = predata->colB;
549
550     // local variables
551     double  *tempr = predata->w;
552     INT i;
553
555     AMG_data *mgl_v = predata->mgl_data_v;
556     AMG_param *amgparam_v = predata->param_v;
557     ITS_param *itparam_v = predata->ITS_param_v;
558
559     dvector rv; rv.row = colA; rv.val = r;
560     dvector zv; zv.row = colA; zv.val = z;
561     dvector rs; rs.row = colB; rs.val = r+colA;
562     dvector zs; zs.row = colB; zs.val = z+colA;
563
565     fasp_darray_cp(col, r, tempr);
566     fasp_darray_set(col, z, 0.0);
567
568     //--------------------
570     //--------------------
571 #if INEXACT
572
573     precond_data pcdata_v;
574     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
575     pcdata_v.max_levels = mgl_v[0].num_levels;
576     pcdata_v.mgl_data = predata->mgl_data_v;
577     precond pc_v; pc_v.data = &pcdata_v;
578     pc_v.fct = fasp_precond_amg;
579
580     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
581
582     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
    restart, 1, itparam_v->print_level);
583 #else
584
585     dCSRmat tmpA;
586     dCSRmat *ptrA = &tmpA;
587     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
588     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
589     fasp_dcsr_free(ptrA);
590
591 #endif
592
593     //--------------------
595     //--------------------
596     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
597
598     //-------------------------
600     //-------------------------
601     ITS_param *itparam_p = predata->ITS_param_p;
602
603 #if INEXACT
604
605     if (itparam_p->precond_type == 1) {
606         precond pc_s;
607         pc_s.data = predata->diag_S;
608         pc_s.fct  = fasp_precond_diag;
609         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
    ->restart, 1, itparam_p->print_level);
610     }
611     else if (itparam_p->precond_type == 2){
613         AMG_data *mgl_p = predata->mgl_data_p;
614         AMG_param *amgparam_p = predata->param_p;
```

```
615
616         precond_data pcdata_p;
617         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
618         pcdata_p.max_levels = mgl_p[0].num_levels;
619         pcdata_p.mgl_data = predata->mgl_data_p;
620         precond pc_p; pc_p.data = &pcdata_p;
621         pc_p.fct = fasp_precond_amg;
622
623         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
624     }
625     else if (itparam_p->precond_type == 4) {
626
627         ILU_data *LU_p = predata->ILU_p;
628
629         precond pc_ilu;
630         pc_ilu.data = LU_p;
631         pc_ilu.fct  = fasp_precond_ilu;
632
633         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
634
635     }
636
637 #else
638
639     fasp_dcsr_trans(predata->S,ptrA);
640     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
641     fasp_dcsr_free(ptrA);
642
643 #endif
644
645     //--------------------
647     //--------------------
648     fasp_blas_dcsr_mxv(predata->Bt, zs.val, rv.val); // rv = B^T zs
649
650     for (i=0;i<colA;i++)
651     {
652         if (predata->diag_A->val[i] > SMALLREAL) rv.val[i]   = rv.val[i]/predata->
      diag_A->val[i]; // rv = D^{-1}rv
653     }
654
655     fasp_blas_darray_axpy (colA, -1.0, rv.val, zv.val); // zu = zu - rv
656
657
658     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
659     //--------------------
661     //--------------------
662     fasp_darray_cp(col, tempr, r);
663
664 }
```

4.10.2.8 fasp_precond_ns_simpler()

```
void fasp_precond_ns_simpler (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Compute rs = rs - B D^{-1} rv

Solve Schur complement

Compute residule

Solve velocity

restore r

Compute residule

Solve Schur complement

Compute zs = zs + deltaS

Compute zu = zu - D^{-1}B^T deltaS

restore r

Definition at line 678 of file PreNavierStokes.c.

```
681 {
682
683     precond_ns_data *predata=(precond_ns_data *)data;
684     const int col = predata->col, colA = predata->colA, colB = predata->colB;
685
686     // local variables
687     double  *tempr = predata->w;
688     INT i;
689
690     dvector *deltaS = predata->sp;
691
693     AMG_data *mgl_v = predata->mgl_data_v;
694     AMG_param *amgparam_v = predata->param_v;
695     ITS_param *itparam_v = predata->ITS_param_v;
696
697     dvector rv; rv.row = colA; rv.val = r;
698     dvector zv; zv.row = colA; zv.val = z;
699     dvector rs; rs.row = colB; rs.val = r+colA;
700     dvector zs; zs.row = colB; zs.val = z+colA;
701
703     fasp_darray_cp(col, r, tempr);
704     fasp_darray_set(col, z, 0.0);
705
706     //--------------------
708     //--------------------
709     for (i=0;i<colA;i++)
710     {
711         if (predata->diag_A->val[i] > SMALLREAL) zv.val[i]   = rv.val[i]/predata->
      diag_A->val[i]; // zv = D^{-1}rv
712     }
713
714     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val); // rs = rs - B zv
715
716     //-------------------------
718     //-------------------------
719     ITS_param *itparam_p = predata->ITS_param_p;
720
721 #if INEXACT
722
723     if (itparam_p->precond_type == 1) {
724         precond pc_s;
725         pc_s.data = predata->diag_S;
726         pc_s.fct  = fasp_precond_diag;
727         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
728     }
729     else if (itparam_p->precond_type == 2){
731         AMG_data *mgl_p = predata->mgl_data_p;
732         AMG_param *amgparam_p = predata->param_p;
733
734         precond_data pcdata_p;
735         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
736         pcdata_p.max_levels = mgl_p[0].num_levels;
737         pcdata_p.mgl_data = predata->mgl_data_p;
738         precond pc_p; pc_p.data = &pcdata_p;
739         pc_p.fct = fasp_precond_amg;
740
741         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
742     }
743     else if (itparam_p->precond_type == 4) {
744
745         ILU_data *LU_p = predata->ILU_p;
746
747         precond pc_ilu;
748         pc_ilu.data = LU_p;
```

```
749         pc_ilu.fct  = fasp_precond_ilu;
750
751         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
752
753     }
754
755 #else
756
757     dCSRmat tmpA;
758     dCSRmat *ptrA = &tmpA;
759     fasp_dcsr_trans(predata->S,ptrA);
760     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
761     fasp_dcsr_free(ptrA);
762
763 #endif
764
765     //--------------------
767     //--------------------
768     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
769
770     //--------------------
772     //--------------------
773 #if INEXACT
774
775     precond_data pcdata_v;
776     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
777     pcdata_v.max_levels = mgl_v[0].num_levels;
778     pcdata_v.mgl_data = predata->mgl_data_v;
779     precond pc_v; pc_v.data = &pcdata_v;
780     pc_v.fct = fasp_precond_amg;
781
782     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
783
784     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
   restart, 1, itparam_v->print_level);
785 #else
786
787     //dCSRmat tmpA;
788     //dCSRmat *ptrA = &tmpA;
789     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
790     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
791     fasp_dcsr_free(ptrA);
792
793 #endif
794
795
796     //--------------------
798     //--------------------
799     fasp_darray_cp(col, tempr, r);
800
801     //--------------------
803     //--------------------
804     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
805
806     //--------------------------
808     //--------------------------
809
810 #if INEXACT
811
812     if (itparam_p->precond_type == 1) {
813         precond pc_s;
814         pc_s.data = predata->diag_S;
815         pc_s.fct  = fasp_precond_diag;
816         fasp_solver_dcsr_pvfgmres(predata->S, &rs, deltaS, &pc_s, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
817     }
818     else if (itparam_p->precond_type == 2){
820         AMG_data *mgl_p = predata->mgl_data_p;
821         AMG_param *amgparam_p = predata->param_p;
822
823         precond_data pcdata_p;
824         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
825         pcdata_p.max_levels = mgl_p[0].num_levels;
826         pcdata_p.mgl_data = predata->mgl_data_p;
827         precond pc_p; pc_p.data = &pcdata_p;
828         pc_p.fct = fasp_precond_amg;
829
830         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, deltaS, &pc_p, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
831     }
832     else if (itparam_p->precond_type == 4) {
833
834         ILU_data *LU_p = predata->ILU_p;
835
836         precond pc_ilu;
837         pc_ilu.data = LU_p;
```

```
838         pc_ilu.fct  = fasp_precond_ilu;
839
840         fasp_solver_dcsr_pvfgmres(predata->S, &rs, deltaS, &pc_ilu, itparam_p->tol,itparam_p->maxit,
    itparam_p->restart, 1, itparam_p->print_level);
841
842     }
843
844 #else
845
846     fasp_dcsr_trans(predata->S,ptrA);
847     fasp_solver_umfpack(ptrA, &rs, deltaS, 0);
848     fasp_dcsr_free(ptrA);
849
850 #endif
851
852     //-------------------
854     //-------------------
855     fasp_blas_darray_axpy(colB, -1.0, deltaS->val, zs.val);
856
857     //-------------------
859     //-------------------
860     fasp_blas_dcsr_mxv(predata->Bt, deltaS->val, rv.val); // rv = B^T deltaS
861
862     for (i=0;i<colA;i++)
863     {
864         if (predata->diag_A->val[i] > SMALLREAL) rv.val[i]   = rv.val[i]/predata->
    diag_A->val[i]; // rv = D^{-1}rv
865     }
866
867     fasp_blas_darray_axpy (colA, -1.0, rv.val, zv.val); // zu = zu - rv
868
869     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
870     //-------------------
872     //-------------------
873     fasp_darray_cp(col, tempr, r);
874
875     // free
876     //fasp_dvec_free (&deltaS);
877
878 }
```

4.10.2.9 fasp_precond_ns_up_btri()

```
void fasp_precond_ns_up_btri (
             REAL * r,
             REAL * z,
             void * data )
```

back up r, setup z;

Solve Schur complement

Compute residule

Solve velocity

prepare AMG preconditioner

restore r

Definition at line 279 of file PreNavierStokes.c.

```
282 {
283     precond_ns_data *predata=(precond_ns_data *)data;
284     const int col = predata->col, colA = predata->colA, colB = predata->colB;
285     //const int maxit = predata->maxit;
286     //double *diagptr=predata->diag_S->val;
287 
288     // local variables
289     double  *tempr = predata->w;
290 
291     dvector rv; rv.row = colA; rv.val = r;
292     dvector zv; zv.row = colA; zv.val = z;
293     dvector rs; rs.row = colB; rs.val = r+colA;
294     dvector zs; zs.row = colB; zs.val = z+colA;
295 
297     fasp_darray_cp(col, r, tempr);
298     fasp_darray_set(col, z, 0.0);
299 
300     //-------------------------
302     //-------------------------
303     ITS_param *itparam_p = predata->ITS_param_p;
304 
305 #if INEXACT
306 
307     if (itparam_p->precond_type == 1) {
308         precond pc_s;
309         pc_s.data = predata->diag_S;
310         pc_s.fct  = fasp_precond_diag;
311         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_s, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
312     }
313     else if (itparam_p->precond_type == 2){
315         AMG_data *mgl_p = predata->mgl_data_p;
316         AMG_param *amgparam_p = predata->param_p;
317 
318         precond_data pcdata_p;
319         fasp_param_amg_to_prec(&pcdata_p,amgparam_p);
320         pcdata_p.max_levels = mgl_p[0].num_levels;
321         pcdata_p.mgl_data = predata->mgl_data_p;
322         precond pc_p; pc_p.data = &pcdata_p;
323         pc_p.fct = fasp_precond_amg;
324 
325         fasp_solver_dcsr_pvfgmres(&mgl_p[0].A, &rs, &zs, &pc_p, itparam_p->tol,itparam_p->maxit, itparam_p
      ->restart, 1, itparam_p->print_level);
326     }
327     else if (itparam_p->precond_type == 4) {
328 
329         ILU_data *LU_p = predata->ILU_p;
330 
331         precond pc_ilu;
332         pc_ilu.data = LU_p;
333         pc_ilu.fct  = fasp_precond_ilu;
334 
335         fasp_solver_dcsr_pvfgmres(predata->S, &rs, &zs, &pc_ilu, itparam_p->tol,itparam_p->maxit,
      itparam_p->restart, 1, itparam_p->print_level);
336 
337     }
338 
339 #else
340 
341     dCSRmat tmpA;
342     dCSRmat *ptrA = &tmpA;
343     fasp_dcsr_trans(predata->S,ptrA);
344     fasp_solver_umfpack(ptrA, &rs, &zs, 0);
345     fasp_dcsr_free(ptrA);
346 
347 #endif
348 
349     //--------------------
351     //--------------------
352     fasp_blas_dcsr_aAxpy(-1.0, predata->Bt, zs.val, rv.val);
353 
354     //--------------------
356     //--------------------
358     AMG_data *mgl_v = predata->mgl_data_v;
359     AMG_param *amgparam_v = predata->param_v;
360     ITS_param *itparam_v = predata->ITS_param_v;
361 
362 #if INEXACT
363 
364     precond_data pcdata_v;
365     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
366     pcdata_v.max_levels = mgl_v[0].num_levels;
367     pcdata_v.mgl_data = predata->mgl_data_v;
368     precond pc_v; pc_v.data = &pcdata_v;
369     pc_v.fct = fasp_precond_amg;
370 
371     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
```

```
      restart, 1, itparam_v->print_level);
372
373 #else
374
375     //dCSRmat tmpA;
376     //dCSRmat *ptrA = &tmpA;
377     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
378     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
379     fasp_dcsr_free(ptrA);
380
381 #endif
382
384     fasp_darray_cp(col, tempr, r);
385 }
```

4.10.2.10 fasp_precond_ns_uzawa()

```
void fasp_precond_ns_uzawa (
             REAL * r,
             REAL * z,
             void * data )
```

prepare AMG preconditioner

back up r, setup z;

Solve velocity

Compute B zv - rs

Compute zs = omega*(-1)*(rs)

restore r

Definition at line 892 of file PreNavierStokes.c.

```
895 {
896
897     precond_ns_data *predata=(precond_ns_data *)data;
898     const int col = predata->col, colA = predata->colA, colB = predata->colB;
899
900     // local variables
901     double  *tempr = predata->w;
902
904     AMG_data *mgl_v = predata->mgl_data_v;
905     AMG_param *amgparam_v = predata->param_v;
906     ITS_param *itparam_v = predata->ITS_param_v;
907
908     dvector rv; rv.row = colA; rv.val = r;
909     dvector zv; zv.row = colA; zv.val = z;
910     dvector rs; rs.row = colB; rs.val = r+colA;
911     dvector zs; zs.row = colB; zs.val = z+colA;
912
914     fasp_darray_cp(col, r, tempr);
915     fasp_darray_set(col, z, 0.0);
916
917     //--------------------
919     //--------------------
920 #if INEXACT
921
922     precond_data pcdata_v;
923     fasp_param_amg_to_prec(&pcdata_v,amgparam_v);
924     pcdata_v.max_levels = mgl_v[0].num_levels;
925     pcdata_v.mgl_data = predata->mgl_data_v;
926     precond pc_v; pc_v.data = &pcdata_v;
927     pc_v.fct = fasp_precond_amg;
928
929     if(itparam_v->print_level > 0)  printf(COLOR_RESET "\n");
930
```

```
931     fasp_solver_dcsr_pvfgmres(&mgl_v[0].A, &rv, &zv, &pc_v, itparam_v->tol, itparam_v->maxit, itparam_v->
      restart, 1, itparam_v->print_level);
932 #else
933
934     dCSRmat tmpA;
935     dCSRmat *ptrA = &tmpA;
936     fasp_dcsr_trans(&mgl_v[0].A,ptrA);
937     fasp_solver_umfpack(ptrA, &rv, &zv, 0);
938     fasp_dcsr_free(ptrA);
939
940 #endif
941
942     //-------------------
944     //-------------------
945     fasp_blas_dcsr_aAxpy(-1.0, predata->B, zv.val, rs.val);
946
947     //-------------------
949     //-------------------
950     REAL omega = -1.0;
951     fasp_blas_darray_axpy(colB, omega, rs.val,zs.val);
952
953     if(itparam_v->print_level > 0)  printf(COLOR_GREEN "\n");
955     fasp_darray_cp(col, tempr, r);
956
957 }
```

4.11 PrePNPStokes.c File Reference

Preconditioners for PNP+Stokes problems.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- void fasp_precond_pnp_stokes_diag (REAL *r, REAL *z, void *data)

  block diagonal preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_lower (REAL *r, REAL *z, void *data)

  block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_upper (REAL *r, REAL *z, void *data)

  block upper triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_diag_inexact (REAL *r, REAL *z, void *data)

  block diagonal preconditioning (3x3 block matrix, each diagonal block is solved inexactly)
- void fasp_precond_pnp_stokes_lower_inexact (REAL *r, REAL *z, void *data)

  block lower triangular preconditioning (3x3 block matrix, each diagonal block is solved exactly)
- void fasp_precond_pnp_stokes_upper_inexact (REAL *r, REAL *z, void *data)

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

4.11.1 Detailed Description

Preconditioners for PNP+Stokes problems.

Note

This file contains Level-4 (Pre) functions.

// TODO: Fix Doxygen. –Chensong

4.11.2 Function Documentation

4.11.2.1 fasp_precond_pnp_stokes_diag()

```
void fasp_precond_pnp_stokes_diag (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 36 of file PrePNPStokes.c.

```
39 {
40
41     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
42     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
43     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
44     dvector *tempr = &(precdata->r);
45
46     const INT N0 = A_pnp_csr->row;
47     const INT N1 = A_stokes_csr->row;
48     const INT N = N0 + N1;
49
50     // back up r, setup z;
51     fasp_darray_cp(N, r, tempr->val);
52     fasp_darray_set(N, z, 0.0);
53
54     // prepare
55 #if  WITH_UMFPACK
56     void **LU_diag = precdata->LU_diag;
57     dvector r0, r1, z0, z1;
58
59     r0.row = N0; z0.row = N0;
60     r1.row = N1; z1.row = N1;
61
62     r0.val = r; r1.val = &(r[N0]);
63     z0.val = z; z1.val = &(z[N0]);
64 #endif
```

```
65
66     // Preconditioning pnp block
67 #if  WITH_UMFPACK
68     /* use UMFPACK direct solver */
69     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
70 #endif
71
72     // Preconditioning A11 block
73 #if  WITH_UMFPACK
74     /* use UMFPACK direct solver */
75     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
76 #endif
77
78     // restore r
79     fasp_darray_cp(N, tempr->val, r);
80
81 }
```

4.11.2.2 fasp_precond_pnp_stokes_diag_inexact()

```
void fasp_precond_pnp_stokes_diag_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 219 of file PrePNPStokes.c.

```
222 {
223
224     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
225     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
226     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
227     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
228     dvector *tempr = &(precdata->r);
229
230     void **LU_diag = precdata->LU_diag;
231     precond_data_bsr *precdata_pnp = precdata->precdata_pnp;
232     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
233
234     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
235     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
236     const INT N = N0 + N1;
237
238     // back up r, setup z;
```

```
239     fasp_darray_cp(N, r, tempr->val);
240     fasp_darray_set(N, z, 0.0);
241
242     // prepare
243     dvector r0, r1, z0, z1;
244
245     r0.row = N0; z0.row = N0;
246     r1.row = N1; z1.row = N1;
247
248     r0.val = r; r1.val = &(r[N0]);
249     z0.val = z; z1.val = &(z[N0]);
250
251     // Preconditioning pnp block
252     //precond prec_pnp;
253
254     //prec_pnp.data = precdata_pnp;
255     //prec_pnp.fct = precdata->pnp_fct;
256
257     //prec_pnp.data = precdata->ILU_pnp;
258     //prec_pnp.fct = fasp_precond_dbsr_ilu;
259
260     //prec_pnp.data = precdata->ILU_pnp;
261     //prec_pnp.fct = fasp_precond_ilu;
262
263     //prec_pnp.data = precdata->diag_pnp;
264     //prec_pnp.fct = fasp_precond_dbsr_diag;
265
266     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
267     fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
268     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
269     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
270
271     // Preconditioning A11 block
272     precond prec_stokes;
273     prec_stokes.data = precdata_stokes;
274     prec_stokes.fct = precdata->stokes_fct;
275
276     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
277
278
279     // restore r
280     fasp_darray_cp(N, tempr->val, r);
281
282 }
```

4.11.2.3 fasp_precond_pnp_stokes_lower()

```
void fasp_precond_pnp_stokes_lower (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 95 of file PrePNPStokes.c.

```
98 {
99
100     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
101     dBLCmat *A = precdata->Abcsr;
102     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
103     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
104
105     dvector *tempr = &(precdata->r);
106
107     const INT N0 = A_pnp_csr->row;
108     const INT N1 = A_stokes_csr->row;
109     const INT N = N0 + N1;
110
111     // back up r, setup z;
112     fasp_darray_cp(N, r, tempr->val);
113     fasp_darray_set(N, z, 0.0);
114
115     // prepare
116     dvector r0, r1, z0, z1;
117
118     r0.row = N0; z0.row = N0;
119     r1.row = N1; z1.row = N1;
120
121     r0.val = r; r1.val = &(r[N0]);
122     z0.val = z; z1.val = &(z[N0]);
123
124     // Preconditioning pnp block
125 #if  WITH_UMFPACK
126     void **LU_diag = precdata->LU_diag;
127     /* use UMFPACK direct solver */
128     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
129 #endif
130
131     // r1 = r1 - A3*z0
132     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[2], z0.val, r1.val);
133
134     // Preconditioning stokes block
135 #if  WITH_UMFPACK
136     /* use UMFPACK direct solver */
137     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
138 #endif
139
140     // restore r
141     fasp_darray_cp(N, tempr->val, r);
142
143 }
```

4.11.2.4 fasp_precond_pnp_stokes_lower_inexact()

```
void fasp_precond_pnp_stokes_lower_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 296 of file PrePNPStokes.c.

```
299 {
300
301     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
302     dBLCmat *A = precdata->Abcsr;
303     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
304     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
305     //dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
306     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
307
308     dvector *tempr = &(precdata->r);
309
310     void **LU_diag = precdata->LU_diag;
311     precond_data_bsr *precdata_pnp= precdata->precdata_pnp;
312     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
313
314     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
315     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
316     const INT N = N0 + N1;
317
318     // back up r, setup z;
319     fasp_darray_cp(N, r, tempr->val);
320     fasp_darray_set(N, z, 0.0);
321
322     // prepare
323     dvector r0, r1, z0, z1;
324
325     r0.row = N0; z0.row = N0;
326     r1.row = N1; z1.row = N1;
327
328     r0.val = r; r1.val = &(r[N0]);
329     z0.val = z; z1.val = &(z[N0]);
330
331     // Preconditioning pnp block
332     //precond prec_pnp;
333
334     //prec_pnp.data = precdata_pnp;
335     //prec_pnp.fct = precdata->pnp_fct;
336
337     //prec_pnp.data = precdata->ILU_pnp;
338     //prec_pnp.fct = fasp_precond_dbsr_ilu;
339
340     //prec_pnp.data = precdata->ILU_pnp;
341     //prec_pnp.fct = fasp_precond_ilu;
342
343     //prec_pnp.data = precdata->diag_pnp;
344     //prec_pnp.fct = fasp_precond_dbsr_diag;
345
346     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
347     fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
348     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
349     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
350
351     // r1 = r1 - A3*z0
352     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[2], z0.val, r1.val);
353
354     // Preconditioning stokes block
355     precond prec_stokes;
356     prec_stokes.data = precdata_stokes;
357     prec_stokes.fct = precdata->stokes_fct;
358
359     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
360
361     // restore r
362     fasp_darray_cp(N, tempr->val, r);
363
364 }
```

4.11.2.5 fasp_precond_pnp_stokes_upper()

```
void fasp_precond_pnp_stokes_upper (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 157 of file PrePNPStokes.c.

```
160 {
161
162     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
163     dBLCmat *A = precdata->Abcsr;
164     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
165     dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
166
167     dvector *tempr = &(precdata->r);
168
169     const INT N0 = A_pnp_csr->row;
170     const INT N1 = A_pnp_csr->row;
171     const INT N = N0 + N1;
172
173     // back up r, setup z;
174     fasp_darray_cp(N, r, tempr->val);
175     fasp_darray_set(N, z, 0.0);
176
177     // prepare
178     dvector r0, r1, z0, z1;
179
180     r0.row = N0; z0.row = N0;
181     r1.row = N1; z1.row = N1;
182
183     r0.val = r; r1.val = &(r[N0]);
184     z0.val = z; z1.val = &(z[N0]);
185
186     // Preconditioning stokes block
187 #if  WITH_UMFPACK
188     void **LU_diag = precdata->LU_diag;
189     /* use UMFPACK direct solver */
190     fasp_umfpack_solve(A_stokes_csr, &r1, &z1, LU_diag[1], 0);
191 #endif
192
193     // r1 = r1 - A5*z2
194     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[1], z1.val, r0.val);
195
196     // Preconditioning pnp block
197 #if  WITH_UMFPACK
198     /* use UMFPACK direct solver */
199     fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
200 #endif
201
202     // restore r
203     fasp_darray_cp(N, tempr->val, r);
204
205 }
```

4.11.2.6 fasp_precond_pnp_stokes_upper_inexact()

```
void fasp_precond_pnp_stokes_upper_inexact (
             REAL * r,
             REAL * z,
             void * data )
```

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

Parameters

r	Pointer to the vector needs preconditioning
z	Pointer to preconditioned vector
data	Pointer to precondition data

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 378 of file PrePNPStokes.c.

```
381 {
382
383     precond_pnp_stokes_data *precdata=(
      precond_pnp_stokes_data *)data;
384     dBLCmat *A = precdata->Abcsr;
385     dCSRmat *A_pnp_csr = precdata->A_pnp_csr;
386     dBSRmat *A_pnp_bsr = precdata->A_pnp_bsr;
387     //dCSRmat *A_stokes_csr = precdata->A_stokes_csr;
388     dBLCmat *A_stokes_bcsr = precdata->A_stokes_bcsr;
389
390     dvector *tempr = &(precdata->r);
391
392     void **LU_diag = precdata->LU_diag;
393     precond_data_bsr *precdata_pnp= precdata->precdata_pnp;
394     precond_ns_data  *precdata_stokes = precdata->precdata_stokes;
395
396     const INT N0 = A_pnp_bsr->ROW*A_pnp_bsr->nb;
397     const INT N1 = A_stokes_bcsr->blocks[0]->row + A_stokes_bcsr->blocks[2]->row;
398     const INT N = N0 + N1;
399
400     // back up r, setup z;
401     fasp_darray_cp(N, r, tempr->val);
402     fasp_darray_set(N, z, 0.0);
403
404     // prepare
405     dvector r0, r1, z0, z1;
406
407     r0.row = N0; z0.row = N0;
408     r1.row = N1; z1.row = N1;
409
410     r0.val = r; r1.val = &(r[N0]);
411     z0.val = z; z1.val = &(z[N0]);
412
413     // Preconditioning stokes block
414     precond prec_stokes;
415     prec_stokes.data = precdata_stokes;
416     prec_stokes.fct = precdata->stokes_fct;
417
418     fasp_solver_dblc_pvfgmres(A_stokes_bcsr, &r1, &z1, &prec_stokes, 1e-3, 100, 100, 1, 0);
419
420     // r1 = r1 - A5*z2
421     fasp_blas_dcsr_aAxpy(-1.0, A->blocks[1], z1.val, r0.val);
422
423     // Preconditioning pnp block
```

```
424     //precond prec_pnp;
425
426     //prec_pnp.data = precdata_pnp;
427     //prec_pnp.fct = precdata->pnp_fct;
428
429     //prec_pnp.data = precdata->ILU_pnp;
430     //prec_pnp.fct = fasp_precond_dbsr_ilu;
431
432     //prec_pnp.data = precdata->ILU_pnp;
433     //prec_pnp.fct = fasp_precond_ilu;
434
435     //prec_pnp.data = precdata->diag_pnp;
436     //prec_pnp.fct = fasp_precond_dbsr_diag;
437
438     //fasp_umfpack_solve(A_pnp_csr, &r0, &z0, LU_diag[0], 0);
439     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, NULL, 1e-3, 50, 50, 1, 1);
440     //fasp_solver_dbsr_pvgmres(A_pnp_bsr, &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
441     fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, NULL, 1e-3, 50, 50, 1, 0);
442     //fasp_solver_dcsr_pvgmres(A->blocks[0], &r0, &z0, &prec_pnp, 1e-3, 100, 100, 1, 1);
443
444     // restore r
445     fasp_darray_cp(N, tempr->val, r);
446
447 }
```

4.12 SolNavierStokes.c File Reference

Iterative solvers for Navier-Stokes matrices (main file)

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- SHORT fasp_solver_dblc_krylov_navier_stokes (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam)

  Solve Ax=b by standard Krylov methods for NS equations.
- SHORT fasp_solver_dblc_krylov_navier_stokes_pmass (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam, dCSRmat *Mp)

  Solve Ax=b by standard Krylov methods for NS equations.
- SHORT fasp_solver_dblc_krylov_navier_stokes_schur_pmass (dBLCmat *Mat, dvector *b, dvector *x, itsolver_ns_param *itparam, AMG_ns_param *amgparam, ILU_param *iluparam, SWZ_param *schparam, dCSRmat *Mp)

  Solve Ax=b by standard Krylov methods for NS equations.

4.12.1 Detailed Description

Iterative solvers for Navier-Stokes matrices (main file)

Note

This file contains Level-5 (Sol) functions. It requires: PreNavierStokes.c

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

// TODO: Fix Doxygen. –Chensong // TODO: Shorten function names. –Chensong

4.12.2 Function Documentation

4.12.2.1 fasp_solver_dblc_krylov_navier_stokes()

```
SHORT fasp_solver_dblc_krylov_navier_stokes (
             dBLCmat * Mat,
             dvector * b,
             dvector * x,
             itsolver_ns_param * itparam,
             AMG_ns_param * amgparam,
             ILU_param * iluparam,
             SWZ_param * schparam )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	pionter to AMG parameters for N-S
iluparam	pionter to ILU parameters
swzparam	pionter to Schwarz parameters

Returns

number of iterations

Author

Lu Wang

Date

03/02/2012

Modified by Xiaozhe Hu on 05/31/2016 Modified by Chensong Zhang on 03/15/2018

Definition at line 132 of file SolNavierStokes.c.

```
139 {
140     // parameters
141     const SHORT PrtLvl       = itparam->print_level;
142     const SHORT precond_type = itparam->precond_type;
143     const INT schwarz_mmsize = schparam->SWZ_mmsize;
144     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
145     const INT schwarz_type   = schparam->SWZ_type;
146 
147 #if DEBUG_MODE > 0
148     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
149 #endif
150 
151     // Navier-Stokes 2 by 2 matrix
152     dCSRmat *A  = Mat->blocks[0];
153     dCSRmat *Bt = Mat->blocks[1];
154     dCSRmat *B  = Mat->blocks[2];
155     dCSRmat *C  = Mat->blocks[3];
156 
157     const INT n = A->row, m = B->row, nnzA = A->nnz;
158 
159     // preconditioner data
160     dCSRmat *M = Mat->blocks[3];
161     dCSRmat S;
162     SWZ_data schwarz_data;
163     dvector diag_A;
164     dvector diag_S;
165     dCSRmat BABt;
166 
167     // local variable
168     clock_t solver_start, solver_end, setup_start, setup_end;
169     REAL solver_duration, setup_duration;
170     SHORT status = FASP_SUCCESS;
171 
172     //------ setup phase ------//
173     setup_start = clock();
174 
175 #if DEBUG_MODE > 0
176     printf("### DEBUG: n = %d, m = %d, nnz = %d\n", n, m, nnzA);
177 #endif
178 
179     //-------------------------//
180     // setup AMG for velocity  //
181     //-------------------------//
182     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
183     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA);
184 
185     if (precond_type > 10) {
186 
187         dCSRmat BtB;
188         fasp_blas_dcsr_mxm(Bt, B, &BtB);
189 
190         REAL gamma = 10;
191         fasp_blas_dcsr_add (A, 1.0, &BtB, gamma, &mgl_v[0].A);
192 
193         fasp_dcsr_free(&BtB);
194 
195     }
196     else {
197         fasp_dcsr_cp(A,&mgl_v[0].A);
198     }
199 
200     mgl_v[0].b = fasp_dvec_create(n);
201     mgl_v[0].x = fasp_dvec_create(n);
202 
203     // setup AMG
204     switch (amgparam->param_v.AMG_type) {
205             case CLASSIC_AMG:
206             fasp_amg_setup_rs(mgl_v, &amgparam->param_v); break;
207             case SA_AMG:
208             fasp_amg_setup_sa(mgl_v, &amgparam->param_v); break;
209             case UA_AMG:
210             fasp_amg_setup_ua(mgl_v, &amgparam->param_v); break;
211             default:
212             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
213             exit(ERROR_INPUT_PAR);
214     }
215 
216     // get diagonal of A
217     fasp_dcsr_getdiag(n, &mgl_v[0].A, &diag_A);
218 
219     //---------------------------//
220     // setup Schur complement S  //
221     //---------------------------//
222 
223     if ( precond_type == 8  || precond_type == 9 ||
224         precond_type == 18 || precond_type == 19 ) {
225 
```

```
226         fasp_blas_dcsr_mxm(B, Bt, &S);
227
228         // change the sign of the BB^T
229         fasp_blas_dcsr_axm(&S, -1.0);
230
231         // make it non-singular
232         INT i,k,j,ibegin,iend;
233
234         for (i=0;i<S.row;++i) {
235             ibegin=S.IA[i]; iend=S.IA[i+1];
236             for (k=ibegin;k<iend;++k) {
237                 j=S.JA[k];
238                 if ( j==i ) {
239                     S.val[k] = S.val[k] + 1e-8; break;
240                 } // end if
241             } // end for k
242         } // end for i
243
244     }
245     else if ( precond_type == 10 || precond_type == 20 ) {
246
247         fasp_blas_dcsr_mxm(B, Bt, &S);
248         fasp_blas_dcsr_rap(B, A, Bt, &BABt);
249
250         // change the sign of the BB^T
251         fasp_blas_dcsr_axm(&S, -1.0);
252
253         // make it non-singular
254         INT i,k,j,ibegin,iend;
255
256         for (i=0;i<S.row;++i) {
257             ibegin=S.IA[i]; iend=S.IA[i+1];
258             for (k=ibegin;k<iend;++k) {
259                 j=S.JA[k];
260                 if ( j==i ) {
261                     S.val[k] = S.val[k] + 1e-8; break;
262                 } // end if
263             } // end for k
264         } // end for i
265
266     }
267     else {
268         get_schur_diagA(B,Bt,A,C,&S);
269     }
270
271     dvector res_p = fasp_dvec_create(m);
272     dvector sol_p = fasp_dvec_create(m);
273
274     AMG_data *mgl_p;
275     ILU_data LU_p;
276
277     if ( itparam->precond_type_p == 1 ) {
278         fasp_dcsr_getdiag(0,&S,&diag_S);
279     }
280     else if ( itparam->precond_type_p == 2 ) {
281         // Setup AMG for Schur Complement
282         dCSRmat *As = &S;
283         const INT nnzS = As->nnz;
284         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
285         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
286         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
287         // setup AMG
288         switch (amgparam->param_p.AMG_type) {
289                 case CLASSIC_AMG:
290                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p); break;
291                 case SA_AMG:
292                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p); break;
293                 case UA_AMG:
294                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p); break;
295                 default:
296                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
297                        amgparam->param_p.AMG_type);
298                 exit(ERROR_INPUT_PAR);
299         }
300     }
301     else if ( itparam->precond_type_p == 4 ) {
302         // setup ILU for Schur Complement
303         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
304         fasp_mem_iludata_check(&LU_p);
305     }
306
307     //--------------------------------------//
308     // Setup itsolver parameter for subblocks
309     //--------------------------------------//
310     ITS_param ITS_param_v;
311     fasp_param_solver_init(&ITS_param_v);
312     ITS_param_v.print_level = itparam->print_level_v;
```

```
313     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
314     ITS_param_v.restart = itparam->pre_restart_v;
315     ITS_param_v.tol = itparam->pre_tol_v;
316     ITS_param_v.maxit = itparam->pre_maxit_v;
317     ITS_param_v.precond_type = itparam->precond_type_v;
318
319     ITS_param ITS_param_p;
320     fasp_param_solver_init(&ITS_param_p);
321     ITS_param_p.print_level = itparam->print_level_p;
322     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
323     ITS_param_p.restart = itparam->pre_restart_p;
324     ITS_param_p.tol = itparam->pre_tol_p;
325     ITS_param_p.maxit = itparam->pre_maxit_p;
326     ITS_param_p.precond_type = itparam->precond_type_p;
327
328     //-------------------------//
329     // setup preconditioner
330     //-------------------------//
331     precond prec;
332     precond_ns_data precdata;
333     prec.data = &precdata;
334
335     precdata.colA = n;
336     precdata.colB = m;
337     precdata.col  = n+m;
338     precdata.M    = M;
339     precdata.B    = B;
340     precdata.Bt   = Bt;
341     precdata.C    = C;
342     precdata.BABt = &BABt;
343
344     precdata.param_v     = &amgparam->param_v;
345     precdata.param_p     = &amgparam->param_p;
346     precdata.ITS_param_v = &ITS_param_v;
347     precdata.ITS_param_p = &ITS_param_p;
348     precdata.mgl_data_v  = mgl_v;
349     precdata.mgl_data_p  = mgl_p;
350     precdata.ILU_p       = &LU_p;
351
352     precdata.max_levels     = mgl_v[0].num_levels;
353     precdata.print_level    = amgparam->param_v.print_level;
354     precdata.maxit          = amgparam->param_v.maxit;
355     precdata.amg_tol        = amgparam->param_v.tol;
356     precdata.cycle_type     = amgparam->param_v.cycle_type;
357     precdata.smoother       = amgparam->param_v.smoother;
358     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
359     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
360     precdata.relaxation     = amgparam->param_v.relaxation;
361     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
362
363     precdata.diag_A = &diag_A;
364     precdata.S = &S;
365     precdata.diag_S = &diag_S;
366     precdata.rp = &res_p;
367     precdata.sp = &sol_p;
368     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
369
370     switch (precond_type) {
371             case 1:
372             prec.fct = fasp_precond_ns_bdiag;
373             break;
374             case 2:
375             prec.fct = fasp_precond_ns_low_btri;
376             break;
377             case 3:
378             prec.fct = fasp_precond_ns_up_btri;
379             break;
380             case 4:
381             prec.fct = fasp_precond_ns_blu;
382             break;
383             case 5:
384             prec.fct = fasp_precond_ns_simple;
385             break;
386             case 6:
387             prec.fct = fasp_precond_ns_simpler;
388             break;
389             case 7:
390             prec.fct = fasp_precond_ns_uzawa;
391             break;
392             case 8:
393             prec.fct = fasp_precond_ns_projection;
394             break;
395             case 9:
396             prec.fct = fasp_precond_ns_DGS;
397             break;
398             case 10:
399             prec.fct = fasp_precond_ns_LSCDGS;
```

```
400             break;
401             case 11:
402             prec.fct = fasp_precond_ns_bdiag;
403             break;
404             case 12:
405             prec.fct = fasp_precond_ns_low_btri;
406             break;
407             case 13:
408             prec.fct = fasp_precond_ns_up_btri;
409             break;
410             case 14:
411             prec.fct = fasp_precond_ns_blu;
412             break;
413             case 15:
414             prec.fct = fasp_precond_ns_simple;
415             break;
416             case 16:
417             prec.fct = fasp_precond_ns_simpler;
418             break;
419             case 17:
420             prec.fct = fasp_precond_ns_uzawa;
421             break;
422             case 18:
423             prec.fct = fasp_precond_ns_projection;
424             break;
425             case 19:
426             prec.fct = fasp_precond_ns_DGS;
427             break;
428             case 20:
429             prec.fct = fasp_precond_ns_LSCDGS;
430             break;
431             default:
432             printf("### ERROR: Unknown preconditioner type!\n");
433             exit(ERROR_SOLVER_PRECTYPE);
434     }
435
436     setup_end = clock();
437
438     if (PrtLvl>0) {
439         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
440         printf("Setup costs %f.\n", setup_duration);
441     }
442
443     //------ solve phase ------//
444     solver_start=clock();
445     //status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
446     status=fasp_ns_solver_itsolver(Mat,b,x,NULL,itparam);
447     solver_end=clock();
448
449     if (PrtLvl>0) {
450         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
451         printf(COLOR_RESET);
452         printf("Solver costs %f seconds.\n", solver_duration);
453         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
454     }
455
456     //FINISHED:
457     // clean up memory
458     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
459     if (itparam->precond_type_p == 1) fasp_dvec_free(&diag_S);
460     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
461
462     fasp_mem_free(precdata.w);
463     fasp_dvec_free(&res_p);
464     fasp_dvec_free(&sol_p);
465     fasp_dcsr_free(&S);
466     if (precond_type == 10 || precond_type == 20) fasp_dcsr_free(&BABt);
467     fasp_dvec_free(&diag_A);
468
469     return status;
470 }
```

4.12.2.2 fasp_solver_dblc_krylov_navier_stokes_pmass()

```
SHORT fasp_solver_dblc_krylov_navier_stokes_pmass (
             dBLCmat * Mat,
             dvector * b,
```

```
                dvector * x,
                itsolver_ns_param * itparam,
                AMG_ns_param * amgparam,
                ILU_param * iluparam,
                SWZ_param * schparam,
                dCSRmat * Mp )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	AMG parameters for NS
iluparam	ILU parameters
schparam	Schwarz parameters
precdata	pionter to preconditioner data for ns
Mp	pointer to dCSRmat of the pressure mass matrix

Returns

number of iterations

Author

Xiaozhe Hu

Date

017/07/2014

Note

In general, this is for purely Stokes problem, NS problem with div-div stablization – Xiaozhe

Definition at line 499 of file SolNavierStokes.c.

```
507 {
508     // parameters
509     const SHORT PrtLvl = itparam->print_level;
510     const SHORT precond_type = itparam->precond_type;
511     const INT schwarz_mmsize = schparam->SWZ_mmsize;
512     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
513     const INT schwarz_type   = schparam->SWZ_type;
514
515     // Navier-Stokes 4 by 4 matrix
516     dCSRmat *A  = Mat->blocks[0];
517     dCSRmat *Bt = Mat->blocks[1];
518     dCSRmat *B  = Mat->blocks[2];
519     dCSRmat *C  = Mat->blocks[3];
520     const INT n = A->row, m = B->row, nnzA = A->nnz;
521
522     // preconditioner data
523     dCSRmat *M = Mat->blocks[3];
524     dCSRmat S,P;
525     SWZ_data schwarz_data;
526     dvector diag_S;
```

```
527
528     // local variable
529     clock_t solver_start, solver_end, setup_start, setup_end;
530     REAL    solver_duration, setup_duration;
531     SHORT   status = FASP_SUCCESS;
532
533 #if DEBUG_MODE > 0
534     printf("### DEBUG: %s ...... [Start]\n", __FUNCTION__);
535 #endif
536
537     //------ setup phase ------//
538     setup_start = clock();
539
540     //-----------------------//
541     // setup AMG for velocity
542     //-----------------------//
543     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
544     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA); fasp_dcsr_cp(A,&mgl_v[0].A);
545     mgl_v[0].b=fasp_dvec_create(n); mgl_v[0].x=fasp_dvec_create(n);
546
547     // setup AMG
548     switch (amgparam->param_v.AMG_type) {
549             case CLASSIC_AMG:
550             fasp_amg_setup_rs(mgl_v, &amgparam->param_v);
551             break;
552             case SA_AMG:
553             fasp_amg_setup_sa(mgl_v, &amgparam->param_v);
554             break;
555             case UA_AMG:
556             fasp_amg_setup_ua(mgl_v, &amgparam->param_v);
557             break;
558             default:
559             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
560             exit(ERROR_INPUT_PAR);
561     }
562
563     //-------------------------//
564     // setup Schur complement S using pressure mass
565     //-------------------------//
566
567     fasp_dcsr_alloc(Mp->row, Mp->col, Mp->nnz, &S);
568     fasp_dcsr_cp(Mp, &S);
569
570     dvector res_p = fasp_dvec_create(m);
571     dvector sol_p = fasp_dvec_create(m);
572
573     AMG_data *mgl_p;
574     ILU_data LU_p;
575
576     if ( itparam->precond_type_p == 1 ) {
577         fasp_dcsr_getdiag(0,&S,&diag_S);
578     }
579     else if ( itparam->precond_type_p == 2 ) {
580         // setup AMG for Schur Complement
581         dCSRmat *As = &S;
582         const INT nnzS = As->nnz;
583         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
584         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
585         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
586         // setup AMG
587         switch ( amgparam->param_p.AMG_type ) {
588                 case CLASSIC_AMG:
589                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p);
590                 break;
591                 case SA_AMG:
592                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p);
593                 break;
594                 case UA_AMG:
595                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p);
596                 break;
597                 default:
598                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
599                        amgparam->param_p.AMG_type);
600                 exit(ERROR_INPUT_PAR);
601         }
602     }
603     else if ( itparam->precond_type_p == 4 ) {
604         // setup ILU for Schur Complement
605         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
606         fasp_mem_iludata_check(&LU_p);
607     }
608
609     //---------------------------------------//
610     // Setup itsolver parameter for subblocks
611     //---------------------------------------//
612     ITS_param ITS_param_v;
613     fasp_param_solver_init(&ITS_param_v);
```

```
614     ITS_param_v.print_level = itparam->print_level_v;
615     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
616     ITS_param_v.restart = itparam->pre_restart_v;
617     ITS_param_v.tol = itparam->pre_tol_v;
618     ITS_param_v.maxit = itparam->pre_maxit_v;
619     ITS_param_v.precond_type = itparam->precond_type_v;
620
621     ITS_param ITS_param_p;
622     fasp_param_solver_init(&ITS_param_p);
623     ITS_param_p.print_level = itparam->print_level_p;
624     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
625     ITS_param_p.restart = itparam->pre_restart_p;
626     ITS_param_p.tol = itparam->pre_tol_p;
627     ITS_param_p.maxit = itparam->pre_maxit_p;
628     ITS_param_p.precond_type = itparam->precond_type_p;
629
630     //-------------------------//
631     // setup preconditioner
632     //-------------------------//
633     precond prec;
634     precond_ns_data precdata;
635     prec.data = &precdata;
636
637     precdata.colA = n;
638     precdata.colB = m;
639     precdata.col  = n+m;
640     precdata.M    = M;
641     precdata.B    = B;
642     precdata.Bt   = Bt;
643     precdata.C    = C;
644
645     precdata.param_v        = &amgparam->param_v;
646     precdata.param_p        = &amgparam->param_p;
647     precdata.ITS_param_v    = &ITS_param_v;
648     precdata.ITS_param_p    = &ITS_param_p;
649     precdata.mgl_data_v     = mgl_v;
650     precdata.mgl_data_p     = mgl_p;
651     precdata.ILU_p          = &LU_p;
652
653     precdata.max_levels     = mgl_v[0].num_levels;
654     precdata.print_level    = amgparam->param_v.print_level;
655     precdata.maxit          = amgparam->param_v.maxit;
656     precdata.amg_tol        = amgparam->param_v.tol;
657     precdata.cycle_type     = amgparam->param_v.cycle_type;
658     precdata.smoother       = amgparam->param_v.smoother;
659     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
660     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
661     precdata.relaxation     = amgparam->param_v.relaxation;
662     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
663
664
665     precdata.S = &S;
666     precdata.diag_S = &diag_S;
667     precdata.rp = &res_p;
668     precdata.sp = &sol_p;
669
670     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
671
672     switch (precond_type) {
673             case 1:
674             prec.fct = fasp_precond_ns_bdiag; break;
675             case 2:
676             prec.fct = fasp_precond_ns_low_btri; break;
677             case 3:
678             prec.fct = fasp_precond_ns_up_btri; break;
679             case 4:
680             prec.fct = fasp_precond_ns_blu; break;
681             default:
682             printf("### ERROR: Unknown preconditioner type!\n");
683             exit(ERROR_SOLVER_PRECTYPE);
684     }
685
686     setup_end = clock();
687
688     if (PrtLvl>0) {
689         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
690         printf("Setup costs %f.\n", setup_duration);
691     }
692
693     //------ solver phase ------//
694     solver_start=clock();
695     status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
696     solver_end=clock();
697
698     if (PrtLvl>0) {
699         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
700         printf(COLOR_RESET);
```

```
701         printf("Solver costs %f seconds.\n", solver_duration);
702         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
703     }
704
705     // clean up memory
706     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
707     if (itparam->precond_type_p == 1) fasp_dvec_free(&diag_S);
708     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
709
710     fasp_mem_free(precdata.w);
711     fasp_dvec_free(&res_p);
712     fasp_dvec_free(&sol_p);
713     fasp_dcsr_free(&S);
714
715     return status;
716 }
```

4.12.2.3 fasp_solver_dblc_krylov_navier_stokes_schur_pmass()

```
SHORT fasp_solver_dblc_krylov_navier_stokes_schur_pmass (
             dBLCmat * Mat,
             dvector * b,
             dvector * x,
             itsolver_ns_param * itparam,
             AMG_ns_param * amgparam,
             ILU_param * iluparam,
             SWZ_param * schparam,
             dCSRmat * Mp )
```

Solve Ax=b by standard Krylov methods for NS equations.

Parameters

A	pointer to the dBLCmat matrix
b	pointer to the dvector of right hand side
x	pointer to the dvector of dofs
itparam	pointer to parameters for iterative solvers
amgparam	AMG parameters for NS
iluparam	ILU parameters
schparam	Schwarz parameters
precdata	piointer to preconditioner data for ns
Mp	pointer to dCSRmat of the pressure mass matrix

Returns

number of iterations

Author

Xiaozhe Hu

Date

017/07/2014

Note

In general, this is for NS problems without div-div stablization and pressure stablization (pressure block is zero), moreover, pressure mass matrix is provided.

Definition at line 747 of file SolNavierStokes.c.

```
755 {
756 #if DEBUG_MODE > 0
757     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
758 #endif
759
760     // parameters
761     const SHORT PrtLvl = itparam->print_level;
762     const SHORT precond_type = itparam->precond_type;
763     const INT schwarz_mmsize = schparam->SWZ_mmsize;
764     const INT schwarz_maxlvl = schparam->SWZ_maxlvl;
765     const INT schwarz_type   = schparam->SWZ_type;
766
767     // Navier-Stokes 4 by 4 matrix
768     dCSRmat *A  = Mat->blocks[0];
769     dCSRmat *Bt = Mat->blocks[1];
770     dCSRmat *B  = Mat->blocks[2];
771     dCSRmat *C  = Mat->blocks[3];
772     const INT n = A->row, m = B->row, nnzA = A->nnz;
773
774     // preconditioner data
775     dCSRmat *M = Mat->blocks[3];
776     dCSRmat S,P;
777     SWZ_data schwarz_data;
778     dvector diag_S;
779
780     // local variable
781     clock_t solver_start, solver_end, setup_start, setup_end;
782     REAL solver_duration, setup_duration;
783     SHORT status=FASP_SUCCESS;
784
785     //------ setup phase ------//
786     setup_start = clock();
787
788     //-----------------------//
789     // setup AMG for velocity
790     //-----------------------//
791
792     AMG_data *mgl_v=fasp_amg_data_create(amgparam->param_v.max_levels);
793
794     mgl_v[0].A=fasp_dcsr_create(n,n,nnzA); fasp_dcsr_cp(A,&mgl_v[0].A);
795     mgl_v[0].b=fasp_dvec_create(n); mgl_v[0].x=fasp_dvec_create(n);
796
797     // setup AMG
798     switch (amgparam->param_v.AMG_type) {
799             case CLASSIC_AMG:
800             fasp_amg_setup_rs(mgl_v, &amgparam->param_v);
801             break;
802             case SA_AMG:
803             fasp_amg_setup_sa(mgl_v, &amgparam->param_v);
804             break;
805             case UA_AMG:
806             fasp_amg_setup_ua(mgl_v, &amgparam->param_v);
807             break;
808             default:
809             printf("### ERROR: Wrong AMG type %d!\n",amgparam->param_v.AMG_type);
810             exit(ERROR_INPUT_PAR);
811     }
812
813     //-------------------------//
814     // setup Schur complement S using pressure mass
815     //-------------------------//
816
817     get_schur_pmass(B, Bt, &mgl_v[0].A, Mp, 1e5, &S);
818     // TODO: 1e5 is a parameter can be tuned, not sure how to tune now -- Xiaozhe
819
820     dvector res_p = fasp_dvec_create(m);
821     dvector sol_p = fasp_dvec_create(m);
822
823     AMG_data *mgl_p;
824     ILU_data LU_p;
825
826     if (itparam->precond_type_p == 1){
827         fasp_dcsr_getdiag(0,&S,&diag_S);
828     }
829     else if (itparam->precond_type_p == 2) {
830         // Setup AMG for Schur Complement
```

```
831         dCSRmat *As = &S;
832         const INT nnzS = As->nnz;
833         mgl_p=fasp_amg_data_create(amgparam->param_p.max_levels);
834         mgl_p[0].A=fasp_dcsr_create(m,m,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
835         mgl_p[0].b=fasp_dvec_create(m); mgl_p[0].x=fasp_dvec_create(m);
836         // setup AMG
837         switch (amgparam->param_p.AMG_type) {
838                 case CLASSIC_AMG:
839                 fasp_amg_setup_rs(mgl_p, &amgparam->param_p);
840                 break;
841                 case SA_AMG:
842                 fasp_amg_setup_sa(mgl_p, &amgparam->param_p);
843                 break;
844                 case UA_AMG:
845                 fasp_amg_setup_ua(mgl_p, &amgparam->param_p);
846                 break;
847                 default:
848                 printf("### ERROR: Wrong AMG type %d for Schur Complement!\n",
849                        amgparam->param_p.AMG_type);
850                 exit(ERROR_INPUT_PAR);
851         }
852     }
853     else if (itparam->precond_type_p == 4) {
854         // setup ILU for Schur Complement
855         fasp_ilu_dcsr_setup(&S, &LU_p, iluparam);
856         fasp_mem_iludata_check(&LU_p);
857     }
858
859     //-----------------------------------//
860     // Setup itsolver parameter for subblocks
861     //-----------------------------------//
862     ITS_param ITS_param_v;
863     fasp_param_solver_init(&ITS_param_v);
864     ITS_param_v.print_level = itparam->print_level_v;
865     ITS_param_v.itsolver_type = itparam->itsolver_type_v;
866     ITS_param_v.restart = itparam->pre_restart_v;
867     ITS_param_v.tol = itparam->pre_tol_v;
868     ITS_param_v.maxit = itparam->pre_maxit_v;
869     ITS_param_v.precond_type = itparam->precond_type_v;
870
871     ITS_param ITS_param_p;
872     fasp_param_solver_init(&ITS_param_p);
873     ITS_param_p.print_level = itparam->print_level_p;
874     ITS_param_p.itsolver_type = itparam->itsolver_type_p;
875     ITS_param_p.restart = itparam->pre_restart_p;
876     ITS_param_p.tol = itparam->pre_tol_p;
877     ITS_param_p.maxit = itparam->pre_maxit_p;
878     ITS_param_p.precond_type = itparam->precond_type_p;
879
880     //-------------------------//
881     // setup preconditioner
882     //-------------------------//
883     precond prec;
884     precond_ns_data precdata;
885     prec.data = &precdata;
886
887     precdata.colA = n;
888     precdata.colB = m;
889     precdata.col  = n+m;
890     precdata.M    = M;
891     precdata.B    = B;
892     precdata.Bt   = Bt;
893     precdata.C    = C;
894
895     precdata.param_v        = &amgparam->param_v;
896     precdata.param_p        = &amgparam->param_p;
897     precdata.ITS_param_v    = &ITS_param_v;
898     precdata.ITS_param_p    = &ITS_param_p;
899     precdata.mgl_data_v     = mgl_v;
900     precdata.mgl_data_p     = mgl_p;
901     precdata.ILU_p          = &LU_p;
902
903     precdata.max_levels     = mgl_v[0].num_levels;
904     precdata.print_level    = amgparam->param_v.print_level;
905     precdata.maxit          = amgparam->param_v.maxit;
906     precdata.amg_tol        = amgparam->param_v.tol;
907     precdata.cycle_type     = amgparam->param_v.cycle_type;
908     precdata.smoother       = amgparam->param_v.smoother;
909     precdata.presmooth_iter = amgparam->param_v.presmooth_iter;
910     precdata.postsmooth_iter= amgparam->param_v.postsmooth_iter;
911     precdata.relaxation     = amgparam->param_v.relaxation;
912     precdata.coarse_scaling = amgparam->param_v.coarse_scaling;
913
914     precdata.S = &S;
915     precdata.diag_S = &diag_S;
916     precdata.rp = &res_p;
917     precdata.sp = &sol_p;
```

```
918
919     precdata.w = (REAL *)fasp_mem_calloc(precdata.col,sizeof(double));
920
921     switch (precond_type) {
922             case 1:
923             prec.fct = fasp_precond_ns_bdiag;
924             break;
925             case 2:
926             prec.fct = fasp_precond_ns_low_btri;
927             break;
928             case 3:
929             prec.fct = fasp_precond_ns_up_btri;
930             break;
931             case 4:
932             prec.fct = fasp_precond_ns_blu;
933             break;
934             default:
935             printf("### ERROR: Unknown preconditioner type!\n");
936             exit(ERROR_SOLVER_PRECTYPE);
937     }
938
939     setup_end = clock();
940
941     if (PrtLvl>0) {
942         setup_duration = (double)(setup_end - setup_start)/(double)(CLOCKS_PER_SEC);
943         printf("Setup costs %f.\n", setup_duration);
944     }
945
946     //------ solver phase ------//
947     solver_start=clock();
948     status=fasp_ns_solver_itsolver(Mat,b,x,&prec,itparam);
949     solver_end=clock();
950
951     if (PrtLvl>0) {
952         solver_duration = (double)(solver_end - solver_start)/(double)(CLOCKS_PER_SEC);
953         printf(COLOR_RESET);
954         printf("Solver costs %f seconds.\n", solver_duration);
955         printf("Total costs %f seconds.\n", setup_duration + solver_duration);
956     }
957
958     //FINISHED:
959     // clean up memory
960     if (mgl_v) fasp_amg_data_free(mgl_v,&amgparam->param_v);
961     if (itparam->precond_type_p == 1){fasp_dvec_free(&diag_S);}
962     if (itparam->precond_type_p == 2) fasp_amg_data_free(mgl_p,&amgparam->param_p);
963
964     fasp_mem_free(precdata.w);
965     fasp_dvec_free(&res_p);
966     fasp_dvec_free(&sol_p);
967     fasp_dcsr_free (&S);
968
969     return status;
970 }
```

4.13 SolPNPStokes.c File Reference

Iterative solvers for PNP-Stokes system (main file)

```
#include <math.h>
#include <time.h>
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- INT fasp_solver_dblc_krylov_pnp_stokes (dBLCmat *A, dvector *b, dvector *x, ITS_param *itparam, ITS↩ _param *itparam_pnp, AMG_param *amgparam_pnp, itsolver_ns_param *itparam_stokes, AMG_ns_param *amgparam_stokes, const int num_velocity, const int num_pressure)

  Solve Ax = b by standard Krylov methods.

4.13.1 Detailed Description

Iterative solvers for PNP-Stokes system (main file)

Note

This file contains Level-5 (Sol) functions. It requires: PreNavierStokes.c and PrePNPStokes.c

// TODO: Fix Doxygen. –Chensong

4.13.2 Function Documentation

4.13.2.1 fasp_solver_dblc_krylov_pnp_stokes()

```
INT fasp_solver_dblc_krylov_pnp_stokes (
             dBLCmat * A,
             dvector * b,
             dvector * x,
             ITS_param * itparam,
             ITS_param * itparam_pnp,
             AMG_param * amgparam_pnp,
             itsolver_ns_param * itparam_stokes,
             AMG_ns_param * amgparam_stokes,
             const int num_velocity,
             const int num_pressure )
```

Solve Ax = b by standard Krylov methods.

Parameters

A	Pointer to the coeff matrix in dBLCmat format
b	Pointer to the right hand side in dvector format
x	Pointer to the approx solution in dvector format
itparam	Pointer to parameters for iterative solvers
amgparam	Pointer to parameters for AMG solvers

Returns

Iteration number if converges; ERROR otherwise.

Author

Xiaozhe Hu

Date

10/12/2016

Definition at line 52 of file SolPNPStokes.c.

```
62 {
63     const SHORT prtlvl = itparam->print_level;
64     const SHORT precond_type = itparam->precond_type;
65
66     INT status = FASP_SUCCESS;
67     REAL setup_start, setup_end, setup_duration;
68     REAL solver_start, solver_end, solver_duration;
69
70     INT m, n, nnz, i, k;
71
72     // local variables
73     dCSRmat A_pnp_csr;
74     dBSRmat A_pnp_bsr;
75
76     dCSRmat A_stokes_csr;
77     dBLCmat A_stokes_bcsr;
78     dCSRmat S;
79     dCSRmat BABt;
80
81     // data for pnp
82     AMG_data_bsr *mgl_pnp=fasp_amg_data_bsr_create(amgparam_pnp->max_levels);
83     ILU_param iluparam_pnp;
84     iluparam_pnp.print_level = amgparam_pnp->print_level;
85     iluparam_pnp.ILU_lfil    = amgparam_pnp->ILU_lfil;
86     iluparam_pnp.ILU_droptol = amgparam_pnp->ILU_droptol;
87     iluparam_pnp.ILU_relax   = amgparam_pnp->ILU_relax;
88     iluparam_pnp.ILU_type    = amgparam_pnp->ILU_type;
89     ILU_data ILU_pnp;
90
91     // data for stokes
92     AMG_data *mgl_v = fasp_amg_data_create(amgparam_stokes->param_v.max_levels);
93     AMG_data *mgl_p;
94     dvector   res_p = fasp_dvec_create(num_pressure);
95     dvector   sol_p = fasp_dvec_create(num_pressure);
96
97 #if WITH_UMFPACK
98     void **LU_diag = (void **)fasp_mem_calloc(2, sizeof(void *));
99 #endif
100
101 #if DEBUG_MODE > 0
102     printf("### DEBUG: %s ..... [Start]\n", __FUNCTION__);
103 #endif
104
105     /* setup preconditioner */
106     fasp_gettime(&setup_start);
107
108     /* diagonal blocks are solved exactly */
109     if ( precond_type > 20 && precond_type < 30 ) {
110 #if WITH_UMFPACK
111         // Need to sort the diagonal blocks for UMFPACK format
112         // pnp block
113         A_pnp_csr = fasp_dcsr_create(A->blocks[0]->row, A->blocks[0]->col, A->blocks[0]->nnz);
114         fasp_dcsr_transz(A->blocks[0], NULL, &A_pnp_csr);
115
116         printf("Factorization for pnp diagonal block: \n");
117         LU_diag[0] = fasp_umfpack_factorize(&A_pnp_csr, prtlvl);
118
119         // stokes block
120         A_stokes_csr = fasp_dcsr_create(A->blocks[3]->row, A->blocks[3]->col, A->blocks[3]->nnz);
121         fasp_dcsr_transz(A->blocks[3], NULL, &A_stokes_csr);
122
123         printf("Factorization for stokes diagonal block: \n");
124         LU_diag[1] = fasp_umfpack_factorize(&A_stokes_csr, prtlvl);
125 #endif
126     }
127
128     /* diagonal blocks are solved inexactly */
129     else if ( precond_type > 30 && precond_type < 40 ) {
130
131         // pnp block
132         {
133             A_pnp_bsr = fasp_format_dcsr_dbsr(A->blocks[0], 3);
134
135             // AMG for pnp
136             /*
137              // initialize A, b, x for mgl_pnp[0]
138              mgl_pnp[0].A = fasp_dbsr_create(A_pnp_bsr.ROW, A_pnp_bsr.COL, A_pnp_bsr.NNZ, A_pnp_bsr.nb,
      A_pnp_bsr.storage_manner);
```

```
139             mgl_pnp[0].b = fasp_dvec_create(mgl_pnp[0].A.ROW*mgl_pnp[0].A.nb);
140             mgl_pnp[0].x = fasp_dvec_create(mgl_pnp[0].A.COL*mgl_pnp[0].A.nb);
141
142             fasp_dbsr_cp(&A_pnp_bsr, &(mgl_pnp[0].A));
143
144             switch (amgparam_pnp->AMG_type) {
145
146             case SA_AMG: // Smoothed Aggregation AMG
147             status = fasp_amg_setup_sa_bsr(mgl_pnp, amgparam_pnp); break;
148
149             default:
150             status = fasp_amg_setup_ua_bsr(mgl_pnp, amgparam_pnp); break;
151
152             }
153
154             if (status < 0) goto FINISHED;
155             */
156
157            // diagonal preconditioner for pnp
158            /*
159             // diag of the pnp matrix
160             fasp_dvec_alloc(A_pnp_bsr.ROW*A_pnp_bsr.nb*A_pnp_bsr.nb, &diag_pnp);
161             for (i = 0; i < A_pnp_bsr.ROW; ++i) {
162             for (k = A_pnp_bsr.IA[i]; k < A_pnp_bsr.IA[i+1]; ++k) {
163             if (A_pnp_bsr.JA[k] == i)
164             memcpy(diag_pnp.val+i*A_pnp_bsr.nb*A_pnp_bsr.nb, A_pnp_bsr.val+k*A_pnp_bsr.nb*A_pnp_bsr.nb,
    A_pnp_bsr.nb*A_pnp_bsr.nb*sizeof(REAL));
165             }
166             }
167
168             for (i=0; i<A_pnp_bsr.ROW; ++i){
169             fasp_blas_smat_inv(&(diag_pnp.val[i*A_pnp_bsr.nb*A_pnp_bsr.nb]), A_pnp_bsr.nb);
170             }
171             */
172
173            // BSR ILU for pnp
174            /*
175             // ILU setup
176             if ( (status = fasp_ilu_dbsr_setup(&A_pnp_bsr, &ILU_pnp, &iluparam_pnp)) < 0 ) goto FINISHED;
177
178             // check iludata
179             if ( (status = fasp_mem_iludata_check(&ILU_pnp)) < 0 ) goto FINISHED;
180             */
181
182            // CSR ILU for pnp
183            /*
184             // ILU setup for whole matrix
185             if ( (status = fasp_ilu_dcsr_setup(A->blocks[0],&ILU_pnp,&iluparam_pnp)) < 0 ) goto FINISHED;
186
187             // check iludata
188             if ( (status = fasp_mem_iludata_check(&ILU_pnp)) < 0 ) goto FINISHED;
189             */
190
191        }
192
193        // stokes block
194        {
195            A_stokes_bcsr.brow = 2;
196            A_stokes_bcsr.bcol = 2;
197            A_stokes_bcsr.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
198            for (i=0; i<4 ;i++) {
199                A_stokes_bcsr.blocks[i] = (dCSRmat *)fasp_mem_calloc(1, sizeof(dCSRmat));
200            }
201
202            ivector velocity_idx;
203            ivector pressure_idx;
204            fasp_ivec_alloc(num_velocity, &velocity_idx);
205            fasp_ivec_alloc(num_pressure, &pressure_idx);
206            for (i=0; i<num_velocity; i++) velocity_idx.val[i] = i;
207            for (i=0; i<num_pressure; i++) pressure_idx.val[i] = num_velocity + i;
208
209            fasp_dcsr_getblk(A->blocks[3], velocity_idx.val, velocity_idx.val, velocity_idx.row,
    velocity_idx.row, A_stokes_bcsr.blocks[0]);
210            fasp_dcsr_getblk(A->blocks[3], velocity_idx.val, pressure_idx.val, velocity_idx.row,
    pressure_idx.row, A_stokes_bcsr.blocks[1]);
211            fasp_dcsr_getblk(A->blocks[3], pressure_idx.val, velocity_idx.val, pressure_idx.row,
    velocity_idx.row, A_stokes_bcsr.blocks[2]);
212            fasp_dcsr_getblk(A->blocks[3], pressure_idx.val, pressure_idx.val, pressure_idx.row,
    pressure_idx.row, A_stokes_bcsr.blocks[3]);
213
214            fasp_ivec_free(&velocity_idx);
215            fasp_ivec_free(&pressure_idx);
216
217
218            // AMG for velocity
219            mgl_v[0].A=fasp_dcsr_create(A_stokes_bcsr.blocks[0]->row,A_stokes_bcsr.blocks[0]->col,
    A_stokes_bcsr.blocks[0]->nnz);
```

```
220             fasp_dcsr_cp(A_stokes_bcsr.blocks[0], &mgl_v[0].A);
221             mgl_v[0].b=fasp_dvec_create(A_stokes_bcsr.blocks[0]->row); mgl_v[0].x=fasp_dvec_create(
    A_stokes_bcsr.blocks[0]->col);
222
223             switch (amgparam_stokes->param_v.AMG_type) {
224                 case CLASSIC_AMG:
225                     fasp_amg_setup_rs(mgl_v, &amgparam_stokes->param_v);
226                     break;
227                 case SA_AMG:
228                     fasp_amg_setup_sa(mgl_v, &amgparam_stokes->param_v);
229                     break;
230                 case UA_AMG:
231                     fasp_amg_setup_ua(mgl_v, &amgparam_stokes->param_v);
232                     break;
233                 default:
234                     printf("Error: Wrong AMG type %d!\n",amgparam_stokes->param_v.AMG_type);
235                     exit(ERROR_INPUT_PAR);
236             }
237
238
239             //-------------------------//
240             // setup Schur complement S
241             //-------------------------//
242             fasp_blas_dcsr_mxm(A_stokes_bcsr.blocks[2], A_stokes_bcsr.blocks[1], &S);
243             fasp_blas_dcsr_rap(A_stokes_bcsr.blocks[2], A_stokes_bcsr.blocks[0], A_stokes_bcsr.blocks[1], &
    BABt);
244
245             // change the sign of the BB^T
246             fasp_blas_dcsr_axm(&S, -1.0);
247
248             // make it non-singular
249             INT k,j,ibegin,iend;
250
251             for (i=0;i<S.row;++i) {
252                 ibegin=S.IA[i]; iend=S.IA[i+1];
253                 for (k=ibegin;k<iend;++k) {
254                     j=S.JA[k];
255                     if ((j-i)==0) {
256                         S.val[k] = S.val[k] + 1e-8; break;
257                     } // end if
258                 } // end for k
259             } // end for i
260
261             dCSRmat *As = &S;
262             const int nnzS = As->nnz;
263             mgl_p=fasp_amg_data_create(amgparam_stokes->param_p.max_levels);
264             mgl_p[0].A=fasp_dcsr_create(num_pressure,num_pressure,nnzS); fasp_dcsr_cp(As,&mgl_p[0].A);
265             mgl_p[0].b=fasp_dvec_create(num_pressure); mgl_p[0].x=fasp_dvec_create(num_pressure);
266             // setup AMG
267             switch (amgparam_stokes->param_p.AMG_type) {
268                 case CLASSIC_AMG:
269                     fasp_amg_setup_rs(mgl_p, &amgparam_stokes->param_p);
270                     break;
271                 case SA_AMG:
272                     fasp_amg_setup_sa(mgl_p, &amgparam_stokes->param_p);
273                     break;
274                 case UA_AMG:
275                     fasp_amg_setup_ua(mgl_p, &amgparam_stokes->param_p);
276                     break;
277                 default:
278                     printf("Error: Wrong AMG type %d for Schur Complement!\n",amgparam_stokes->param_p.
    AMG_type);
279                     exit(ERROR_INPUT_PAR);
280             }
281
282         }
283
284     }
285
286     else {
287         fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
288     }
289
290     // generate data for preconditioners
291     // data for pnp
292     precond_data_bsr precdata_pnp;
293     precdata_pnp.print_level = amgparam_pnp->print_level;
294     precdata_pnp.maxit = amgparam_pnp->maxit;
295     precdata_pnp.tol = amgparam_pnp->tol;
296     precdata_pnp.cycle_type = amgparam_pnp->cycle_type;
297     precdata_pnp.smoother = amgparam_pnp->smoother;
298     precdata_pnp.presmooth_iter = amgparam_pnp->presmooth_iter;
299     precdata_pnp.postsmooth_iter = amgparam_pnp->postsmooth_iter;
300     precdata_pnp.coarsening_type = amgparam_pnp->coarsening_type;
301     precdata_pnp.relaxation = amgparam_pnp->relaxation;
302     precdata_pnp.coarse_scaling = amgparam_pnp->coarse_scaling;
303     precdata_pnp.amli_degree = amgparam_pnp->amli_degree;
```

```
304     precdata_pnp.amli_coef = amgparam_pnp->amli_coef;
305     precdata_pnp.tentative_smooth = amgparam_pnp->tentative_smooth;
306     precdata_pnp.max_levels = mgl_pnp[0].num_levels;
307     precdata_pnp.mgl_data = mgl_pnp;
308     precdata_pnp.A = &A_pnp_bsr;
309
310     // data for stokes
311     // Setup itsolver parameters
312     ITS_param ITS_param_v;
313     fasp_param_solver_init(&ITS_param_v);
314     ITS_param_v.print_level = itparam_stokes->print_level_v;
315     ITS_param_v.itsolver_type = itparam_stokes->itsolver_type_v;
316     ITS_param_v.restart = itparam_stokes->pre_restart_v;
317     ITS_param_v.tol = itparam_stokes->pre_tol_v;
318     ITS_param_v.maxit = itparam_stokes->pre_maxit_v;
319     ITS_param_v.precond_type = itparam_stokes->precond_type_v;
320
321     ITS_param ITS_param_p;
322     fasp_param_solver_init(&ITS_param_p);
323     ITS_param_p.print_level = itparam_stokes->print_level_p;
324     ITS_param_p.itsolver_type = itparam_stokes->itsolver_type_p;
325     ITS_param_p.restart = itparam_stokes->pre_restart_p;
326     ITS_param_p.tol = itparam_stokes->pre_tol_p;
327     ITS_param_p.maxit = itparam_stokes->pre_maxit_p;
328     ITS_param_p.precond_type = itparam_stokes->precond_type_p;
329
330     // data for stokes
331     precond_ns_data  precdata_stokes;
332     if ( precond_type > 30 && precond_type < 40 ) {
333         precdata_stokes.colA = A_stokes_bcsr.blocks[0]->row;
334         precdata_stokes.colB = A_stokes_bcsr.blocks[2]->row;
335         precdata_stokes.col  = A_stokes_bcsr.blocks[0]->row + A_stokes_bcsr.blocks[2]->row;
336         precdata_stokes.B  = A_stokes_bcsr.blocks[2];
337         precdata_stokes.Bt = A_stokes_bcsr.blocks[1];
338         precdata_stokes.C = A_stokes_bcsr.blocks[3];
339         precdata_stokes.BABt  = &BABt;
340     }
341
342     precdata_stokes.param_v         = &amgparam_stokes->param_v;
343     precdata_stokes.param_p         = &amgparam_stokes->param_p;
344     precdata_stokes.ITS_param_v     = &ITS_param_v;
345     precdata_stokes.ITS_param_p     = &ITS_param_p;
346     precdata_stokes.mgl_data_v      = mgl_v;
347     precdata_stokes.mgl_data_p      = mgl_p;
348
349     precdata_stokes.max_levels      = mgl_v[0].num_levels;
350     precdata_stokes.print_level     = amgparam_stokes->param_v.print_level;
351     precdata_stokes.maxit           = amgparam_stokes->param_v.maxit;
352     precdata_stokes.amg_tol         = amgparam_stokes->param_v.tol;
353     precdata_stokes.cycle_type      = amgparam_stokes->param_v.cycle_type;
354     precdata_stokes.smoother        = amgparam_stokes->param_v.smoother;
355     precdata_stokes.presmooth_iter  = amgparam_stokes->param_v.presmooth_iter;
356     precdata_stokes.postsmooth_iter = amgparam_stokes->param_v.postsmooth_iter;
357     precdata_stokes.relaxation      = amgparam_stokes->param_v.relaxation;
358     precdata_stokes.coarse_scaling  = amgparam_stokes->param_v.coarse_scaling;
359
360     precdata_stokes.S  = &S;
361     precdata_stokes.rp = &res_p;
362     precdata_stokes.sp = &sol_p;
363
364     precdata_stokes.w = (double *)fasp_mem_calloc(precdata_stokes.col,sizeof(double));
365
366     // data for overall
367     precond_pnp_stokes_data precdata;
368     precdata.Abcsr = A;
369
370 #if WITH_UMFPACK
371     // LU if exact solve
372     precdata.LU_diag = LU_diag;
373 #endif
374
375     // pnp part
376     precdata.A_pnp_csr = &A_pnp_csr;
377     precdata.A_pnp_bsr = &A_pnp_bsr;
378     precdata.precdata_pnp = &precdata_pnp;
379     precdata.pnp_fct = fasp_precond_dbsr_amg;
380     precdata.ILU_pnp = &ILU_pnp;
381
382     // stokes part
383     precdata.A_stokes_csr = &A_stokes_csr;
384     precdata.A_stokes_bcsr = &A_stokes_bcsr;
385     precdata.precdata_stokes = &precdata_stokes;
386     precdata.stokes_fct = fasp_precond_ns_LSCDGS;
387
388     precdata.r = fasp_dvec_create(b->row);
389
390     precond prec; prec.data = &precdata;
```

```
391
392     switch (precond_type)
393     {
394         case 21:
395             prec.fct = fasp_precond_pnp_stokes_diag;
396             break;
397
398         case 22:
399             prec.fct = fasp_precond_pnp_stokes_lower;
400             break;
401
402         case 23:
403             prec.fct = fasp_precond_pnp_stokes_upper;
404             break;
405
406         case 31:
407             prec.fct = fasp_precond_pnp_stokes_diag_inexact;
408             break;
409
410         case 32:
411             prec.fct = fasp_precond_pnp_stokes_lower_inexact;
412             break;
413
414         case 33:
415             prec.fct = fasp_precond_pnp_stokes_upper_inexact;
416             break;
417
418         default:
419             fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
420             break;
421     }
422
423     if ( prtlvl >= PRINT_MIN ) {
424         fasp_gettime(&setup_end);
425         setup_duration = setup_end - setup_start;
426         fasp_cputime("Setup totally", setup_duration);
427     }
428
429
430     // solver part
431     fasp_gettime(&solver_start);
432
433     status=fasp_solver_dblc_itsolver(A,b,x, &prec,itparam);
434
435     fasp_gettime(&solver_end);
436
437     solver_duration = solver_end - solver_start;
438
439     if ( prtlvl >= PRINT_MIN )
440         fasp_cputime("Krylov method totally", solver_duration);
441
442 FINISHED:
443
444     // clean
445     /* diagonal blocks are solved exactly */
446     if ( precond_type > 20 && precond_type < 30 ) {
447 #if WITH_UMFPACK
448         for (i=0; i<2; i++) fasp_umfpack_free_numeric(LU_diag[i]);
449
450         fasp_dcsr_free(&A_pnp_csr);
451         fasp_dcsr_free(&A_stokes_csr);
452
453         fasp_dvec_free(&precdata.r);
454 #endif
455     }
456     /* diagonal blocks are solved by AMG */
457     else if (precond_type > 30 && precond_type < 40) {
458 #if WITH_UMFPACK
459         for (i=0; i<2; i++) fasp_umfpack_free_numeric(LU_diag[i]);
460 #endif
461         fasp_dbsr_free(&A_pnp_bsr);
462         fasp_amg_data_bsr_free(mgl_pnp);
463         //if (&ILU_pnp) fasp_ilu_data_free(&ILU_pnp);
464
465         fasp_dblc_free(&A_stokes_bcsr);
466         fasp_dcsr_free(&S);
467         fasp_dcsr_free(&BABt);
468         fasp_amg_data_free(mgl_v, &amgparam_stokes->param_v);
469         fasp_amg_data_free(mgl_p, &amgparam_stokes->param_p);
470         fasp_dvec_free(&res_p);
471         fasp_dvec_free(&sol_p);
472         fasp_mem_free(precdata_stokes.w);
473
474         fasp_dvec_free(&precdata.r);
475
476     }
477
```

```
478     else {
479         fasp_chkerr(ERROR_SOLVER_PRECTYPE, __FUNCTION__);
480     }
481
482 #if DEBUG_MODE > 0
483     printf("### DEBUG: %s ..... [Finish]\n", __FUNCTION__);
484 #endif
485
486     return status;
487 }
```

4.14 SolWrapper.c File Reference

Wrappers for accessing functions for advanced users.

```
#include "fasp.h"
#include "fasp_functs.h"
#include "fasp4ns.h"
#include "fasp4ns_functs.h"
```

Functions

- void fasp_fwrapper_krylov_navier_stokes_nsym_ (INT *nA, INT *nnzA, INT *ia, INT *ja, REAL *aval, INT *nB, INT *mB, INT *nnzB, INT *ib, INT *jb, REAL *bval, INT *nC, INT *mC, INT *nnzC, INT *ic, INT *jc, REAL *cval, REAL *b, REAL *u)

  Solve [A B; C O] u = b by Krylov method with block preconditioners.
- void fasp_fwrapper_krylov_navier_stokes_sym_ (INT *nA, INT *nnzA, INT *ia, INT *ja, REAL *aval, INT *nB, INT *nnzB, INT *ib, INT *jb, REAL *bval, INT *nC, INT *nnzC, INT *ic, INT *jc, REAL *cval, REAL *b, REAL *u)

  Solve [A B'; B C] u = b by Krylov method with block preconditioners.

4.14.1 Detailed Description

Wrappers for accessing functions for advanced users.

Note

This file contains Level-5 (Sol) functions. It requires: AuxInput.c, AuxParam.c, and SolNavierStokes.c

// TODO: Fix Doxygen. –Chensong

4.14.2 Function Documentation

4.14.2.1 fasp_fwrapper_krylov_navier_stokes_nsym_()

```
void fasp_fwrapper_krylov_navier_stokes_nsym_ (
             INT * nA,
             INT * nnzA,
             INT * ia,
             INT * ja,
             REAL * aval,
             INT * nB,
             INT * mB,
             INT * nnzB,
             INT * ib,
             INT * jb,
             REAL * bval,
             INT * nC,
             INT * mC,
             INT * nnzC,
             INT * ic,
             INT * jc,
             REAL * cval,
             REAL * b,
             REAL * u )
```

Solve [A B; C O] u = b by Krylov method with block preconditioners.

Parameters

nA	num of rows/cols of A
nnzA	num of nonzeros of A
ia	IA of A in CSR format
ja	JA of A in CSR format
aval	VAL of A in CSR format
nB	num of rows of B
mB	num of cols of B
nnzB	num of nonzeros of B
ib	IA of B in CSR format
jb	JA of B in CSR format
bval	VAL of B in CSR format
nC	num of rows of C
mC	num of cols of C
nnzC	num of nonzeros of C
ic	IA of C in CSR format
jc	JA of C in CSR format
cval	VAL of C in CSR format
b	rhs vector
u	solution vector

Author

Lu Wang

Date

03/20/2014

Modified by Chensong Zhang on 03/16/2018 Step 0. Read input parameters

Definition at line 61 of file SolWrapper.c.

```
80 {
81     dBLCmat A; // coefficient matrix
82     dCSRmat matA11, matA21, matA12, matA22;
83     dvector rhs, sol; // right-hand-side, solution
84     precond_ns_param psparam; // parameters for ns precond
85     precond_ns_data  psdata; // data for ns precond
86     int i,flag;
87 
88     char *inputfile = "ini/ns.dat";
89     input_ns_param     inparam;  // parameters from input files
90     itsolver_ns_param  itparam;  // parameters for itsolver
91     AMG_ns_param       amgparam; // parameters for AMG
92     ILU_param          iluparam; // parameters for ILU
93     SWZ_param          swzparam; // parameters for Schwarz
94 
96     fasp_ns_param_input(inputfile, &inparam);
97     fasp_ns_param_init(&inparam, &itparam, &amgparam, &iluparam, &swzparam);
98 
99     // Set local parameters
100     const int print_level   = inparam.print_level;
101     const int problem_num   = inparam.problem_num;
102     const int itsolver_type = inparam.solver_type;
103     const int precond_type  = inparam.precond_type;
104 
105 #if DEBUG_MODE > 0
106     printf("### DEBUG: nA = %d\n", *nA);
107     printf("### DEBUG: nB = %d, mB = %d\n", *nB, *mB);
108     printf("### DEBUG: nC = %d, mc = %d\n", *nC, *mC);
109 #endif
110 
111     // initialize dBLCmat pointer
112     A.brow = 2; A.bcol = 2;
113     A.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
114     if ( A.blocks == NULL ) {
115         printf("### ERROR: Cannot allocate memory %s!\n", __FUNCTION__);
116         exit(ERROR_ALLOC_MEM);
117     }
118     A.blocks[0] = &matA11;
119     A.blocks[1] = &matA12;
120     A.blocks[2] = &matA21;
121     A.blocks[3] = &matA22;
122 
123     // initialize matrix
124     matA11.row = *nA; matA11.col = *nA; matA11.nnz = *nnzA;
125     matA11.IA  = ia;  matA11.JA  = ja;  matA11.val = aval;
126 
127     matA12.row = *nB; matA12.col = *mB; matA12.nnz = *nnzB;
128     matA12.IA  = ib;  matA12.JA  = jb;  matA12.val = bval;
129 
130     matA21.row = *nC; matA21.col = *mC; matA21.nnz = *nnzC;
131     matA21.IA  = ic;  matA21.JA  = jc;  matA21.val = cval;
132 
133     // generate an empty matrix
134     fasp_dcsr_alloc(*nC,*nC,1,&matA22);
135 
136     // shift the index to start from 0 (for C routines)
137     for ( i=0; i<matA11.row+1; i++ ) matA11.IA[i]--;
138     for ( i=0; i<matA12.row+1; i++ ) matA12.IA[i]--;
139     for ( i=0; i<matA21.row+1; i++ ) matA21.IA[i]--;
140     for ( i=0; i<matA11.nnz;   i++ ) matA11.JA[i]--;
141     for ( i=0; i<matA12.nnz;   i++ ) matA12.JA[i]--;
142     for ( i=0; i<matA21.nnz;   i++ ) matA21.JA[i]--;
143 
144     // initialize rhs and sol vectors
145     rhs.row = *nA+*nC; rhs.val = b;
146     sol.row = *nA+*nC; sol.val = u;
147 
148     if (print_level>0) {
149         printf("Max it num = %d\n", inparam.itsolver_maxit);
150         printf("Tolerance  = %e\n", inparam.itsolver_tol);
151     }
152 
153     flag = fasp_solver_dblc_krylov_navier_stokes(&A, &rhs, &sol, &
    itparam,
154                                                  &amgparam, &iluparam, &swzparam);
155 }
```

4.14.2.2 fasp_fwrapper_krylov_navier_stokes_sym_()

```
void fasp_fwrapper_krylov_navier_stokes_sym_ (
             INT * nA,
             INT * nnzA,
             INT * ia,
             INT * ja,
             REAL * aval,
             INT * nB,
             INT * nnzB,
             INT * ib,
             INT * jb,
             REAL * bval,
             INT * nC,
             INT * nnzC,
             INT * ic,
             INT * jc,
             REAL * cval,
             REAL * b,
             REAL * u )
```

Solve [A B'; B C] u = b by Krylov method with block preconditioners.

Parameters

nA	num of cols of A
nnzA	num of nonzeros of A
ia	IA of A in CSR format
ja	JA of A in CSR format
aval	VAL of A in CSR format
nB	num of cols of B
nnzB	num of nonzeros of B
ib	IA of B in CSR format
jb	JA of B in CSR format
bval	VAL of B in CSR format
nC	num of cols of C
nnzC	num of nonzeros of C
ic	IA of C in CSR format
jc	JA of C in CSR format
cval	VAL of C in CSR format
b	rhs vector
u	solution vector

Author

Lu Wang

Date

03/14/2012

Modified by Chensong Zhang on 03/13/2018 Step 0. Read input parameters

Definition at line 189 of file SolWrapper.c.

```
206 {
207     dBLCmat A; // coefficient matrix
208     dCSRmat matA11, matA21, matA12, matA22;
209     dvector rhs, sol; // right-hand-side, solution
210     precond_ns_param psparam; // parameters for ns precond
211     precond_ns_data  psdata; // data for ns precond
212     int i, flag;
213 
215     char *inputfile = "ini/ns.dat";
216     input_ns_param     inparam;  // parameters from input files
217     itsolver_ns_param  itparam;  // parameters for itsolver
218     AMG_ns_param       amgparam; // parameters for AMG
219     ILU_param          iluparam; // parameters for ILU
220     SWZ_param          swzparam; // parameters for Schwarz
221 
222     fasp_ns_param_input(inputfile, &inparam);
223     fasp_ns_param_init(&inparam, &itparam, &amgparam, &iluparam, &swzparam);
224 
225     // set local parameters
226     const int print_level   = inparam.print_level;
227     const int problem_num   = inparam.problem_num;
228     const int itsolver_type = inparam.solver_type;
229     const int precond_type  = inparam.precond_type;
230 
231 #if DEBUG_MODE > 0
232     printf("### DEBUG: nA = %d, nB = %d, nC = %d\n", *nA, *nB, *nC);
233 #endif
234 
235     // initialize dBLCmat pointer
236     A.brow = 2; A.bcol = 2;
237     A.blocks = (dCSRmat **)calloc(4, sizeof(dCSRmat *));
238     if ( A.blocks == NULL ) {
239         printf("### ERROR: Cannot allocate memory %s!\n", __FUNCTION__);
240         exit(ERROR_ALLOC_MEM);
241     }
242     A.blocks[0] = &matA11;
243     A.blocks[1] = &matA12;
244     A.blocks[2] = &matA21;
245     A.blocks[3] = &matA22;
246 
247     // initialize matrix
248     matA11.row = *nA; matA11.col = *nA; matA11.nnz = *nnzA;
249     matA11.IA  = ia;  matA11.JA  = ja;  matA11.val = aval;
250 
251     matA21.row = *nB; matA21.col = *nA; matA21.nnz = *nnzB;
252     matA21.IA  = ib;  matA21.JA  = jb;  matA21.val = bval;
253 
254     matA22.row = *nC; matA22.col = *nC; matA22.nnz = *nnzC;
255     matA22.IA  = ic;  matA22.JA  = jc;  matA22.val = cval;
256 
257     // shift the index to start from 0 (for C routines)
258     for ( i=0; i<matA11.row+1; i++ ) matA11.IA[i]--;
259     for ( i=0; i<matA21.row+1; i++ ) matA21.IA[i]--;
260     for ( i=0; i<matA22.row+1; i++ ) matA22.IA[i]--;
261     for ( i=0; i<matA11.nnz;   i++ ) matA11.JA[i]--;
262     for ( i=0; i<matA21.nnz;   i++ ) matA21.JA[i]--;
263     for ( i=0; i<matA22.nnz;   i++ ) matA22.JA[i]--;
264 
265     // get transform of B
266     fasp_dcsr_trans(&matA21, &matA12);
267 
268     rhs.row = *nA + *nB; rhs.val = b;
269     sol.row = *nA + *nB; sol.val = u;
270 
271     if (print_level>0) {
272         printf("Max it num = %d\n", inparam.itsolver_maxit);
273         printf("Tolerance  = %e\n", inparam.itsolver_tol);
274     }
275 
276     flag = fasp_solver_dblc_krylov_navier_stokes(&A, &rhs, &sol, &
    itparam,
277                                                  &amgparam, &iluparam, &swzparam);
278 }
```

Index