FASP++

0.4.0 Feb/16/2020

Generated by Doxygen 1.8.17

Chapter 1

Introduction

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

The FASPxx is a C++ package designed for developing parallel iterative solvers and preconditioners for PDEs and systems of PDEs. The main components of the package are standard Krylov methods, algebraic multigrid methods, geometric multigrid methods, Schwarz methods, and incomplete factorization methods.

Chapter 2

How to obtain FASP++

TBA

Chapter 3

Building and Installation

This is a simple instruction on building and testing. There is a top level cmake for configuration and building of the FASPxx shared library and the test programs suite. You can use a cmake-style way to compile the package; see https://cmake.org on how to use cmake for your own operating system. To compile, you alos need a C++ compiler.

```
$ mkdir Build; cd Build; cmake ..
```

```
$ make
```

Chapter 4

Developers

Project coordinator:

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

Current active developers (in alphabetic order):

- Fan, Ronghong (AMSS, Chinese Academy of Sciences, China)
- Zhang, Kailei (AMSS, Chinese Academy of Sciences, China)

Chapter 5

Doxygen

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

http://www.doxygen.org

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

Chapter 6

Hierarchical Index

6.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 7

Class Index

7.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 8

File Index

8.1 File List

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

Chapter 9

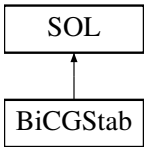
Class Documentation

9.1 BiCGStab Class Reference

Preconditioned bi-conjugate gradient stabilized method.

`#include <BiCGStab.hxx>`

Inheritance diagram for BiCGStab:



Public Member Functions

- BiCGStab ()
  Default constructor.
- ~BiCGStab ()=default
  Default destructor.
- FaspRetCode Setup (const LOP &A) override
  Setup the BiCGStab method.
- void Clean () override
  Clean up CG data allocated during Setup.
- FaspRetCode Solve (const VEC &b, VEC &x) override
  Solve Ax=b using the BiCGStab method.

Additional Inherited Members

9.1.1 Detailed Description

Preconditioned bi-conjugate gradient stabilized method.

9.1.2 Member Function Documentation

9.1.2.1 Clean()

```
void BiCGStab::Clean ( )  [override], [virtual]
```

Clean up CG data allocated during Setup.

Release additional memory allocated for CG.

Reimplemented from SOL.

```
52 {
53     // Nothing is needed for the moment!
54 }
```

9.1.2.2 Setup()

```
FaspRetCode BiCGStab::Setup (
             const LOP & A )  [override], [virtual]
```

Setup the BiCGStab method.

Allocate memory, assign param to this->param.

Reimplemented from SOL.

```
17 {
18     const INT len = A.GetColSize();
19
20     // Allocate memory for temporary vectors
21     try {
22         r0star.SetValues(len, 0.0);
23         tmp.SetValues(len, 0.0);
24         apj.SetValues(len, 0.0);
25         asj.SetValues(len, 0.0);
26         pj.SetValues(len, 0.0);
27         rj.SetValues(len, 0.0);
28         sj.SetValues(len, 0.0);
29         ptmp.SetValues(len, 0.0);
30         stmp.SetValues(len, 0.0);
31         ms.SetValues(len, 0.0);
32         mp.SetValues(len, 0.0);
33         safe.SetValues(len, 0.0);
34     } catch (std::bad_alloc &ex) {
35         return FaspRetCode::ERROR_ALLOC_MEM;
36     }
37
38     // Set method type
39     SetSolType(SOLType::BICGSTAB);
40
41     // Setup the coefficient matrix
42     this->A = &A;
43
44     // Print used parameters
45     if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);
46
47     return FaspRetCode::SUCCESS;
48 }
```

References SOL::A, BICGSTAB, ERROR_ALLOC_MEM, LOP::GetColSize(), SOL::params, SOL::SetSolType(), VEC::SetValues(), and SOLParams::verbose.

9.1.2.3 Solve()

```
FaspRetCode BiCGStab::Solve (
              const VEC & b,
              VEC & x )  [override], [virtual]
```

Solve Ax=b using the BiCGStab method.

Using the Preconditioned Bi-Conjugate Gradient Stabilized method.

Reimplemented from SOL.

```
58 {
59     if ( params.verbose > PRINT_NONE ) std::cout << "Use BiCGStab to solve Ax=b ...\n";
60
61     // Check whether vector space sizes match
62     if ( x.GetSize() != A->GetColSize() || b.GetSize() != A->GetRowSize()
63                                         || A->GetRowSize() != A->GetColSize() )
64         return FaspRetCode::ERROR_NONMATCH_SIZE;
65
66     FaspRetCode errorCode = FaspRetCode::SUCCESS;
67
68     // Declaration and definition of local variables
69     const INT len = b.GetSize();
70     const int maxStag = MAX_STAG_NUM; // maximum number of stagnation before quit
71     const double solStagTol = 1e-4 * params.relTol; // solution stagnation tolerance
72
73     int stagStep = 0, moreStep = 0;
74     double resAbs = 1.0, resRel = 1.0, denAbs = 1.0, ratio = 0.0, resAbsOld = 1.0;
75     double alpha, beta, rjr0star, rjr0startmp, omega, tmp12;
76
77     PrintHead();
78
79     // Initialize iterative method
80     numIter = 0;
81     A->Apply(x, this->tmp); // A * x -> tmp
82     this->rj.WAXPBY(1.0, b, -1.0, this->tmp);
83
84     // Prepare for the main loop
85     this->r0star = this->rj;  // r0_{*} = r0c
86     this->pj     = this->rj;  // p0 = r0
87
88     // Main BiCGStab loop
89     while ( numIter < params.maxIter ) {
90
91         // Start from minIter instead of 0
92         if ( numIter == params.minIter ) {
93             resAbs = rj.Norm2();
94             denAbs = (CLOSE_ZERO > resAbs) ? CLOSE_ZERO : resAbs;
95             resRel = resAbs / denAbs;
96             if ( resRel < params.relTol || resAbs < params.absTol ) break;
97         }
98
99         if ( numIter >= params.minIter ) PrintInfo(numIter, resRel, resAbs, ratio);
100
101          //-------------------------------------------
102          // BiCGStab iteration starts from here
103          //-------------------------------------------
104
105          ++numIter; // iteration count
106
107          /* alpha_{j} = (rj,r0star)/(P * A * pj,r0star) */
108          rjr0star = this->rj.Dot(this->r0star);
109
110          /* main computational work */
111          A->Apply(this->pj, this->apj);
112          ptmp.SetValues(len,0.0);
113          pc->Solve(this->apj, this->ptmp);
114
115          tmp12 = this->ptmp.Dot(this->r0star);
116          if ( fabs(tmp12) > 1e-40 ) alpha = rjr0star / tmp12;
117          else {
118              FASPXX_WARNING("Divided by zero!") // Possible breakdown
119              errorCode = FaspRetCode::ERROR_DIVIDE_ZERO;
120              break;
121          }
122
123          // sj = rj - alpha_{j} * P * A * p_{j}
124          this->sj.WAXPBY(1.0, this->rj, -alpha, this->ptmp);
125
126          // omega_j = (P * A * sj,sj)/(P * A * sj,P * A * sj)
127          A->Apply(this->sj, this->asj);
```

```
128         stmp.SetValues(len,0.0);
129         pc->Solve(this->asj, this->stmp);
130         omega = this->stmp.Dot(this->sj) / this->stmp.Dot(this->stmp);
131
132         /* Update solution and residual */
133         // x_{j+1} = x_{j} + alpha_{j} * P * pj + omega_j * P * s_{j}
134         mp.SetValues(len,0.0);
135         pc->Solve(this->pj, this->mp);
136         ms.SetValues(len,0.0);
137         pc->Solve(this->sj, this->ms);
138         this->tmp.WAXPBY(alpha, this->mp, omega, this->ms);
139         x.XPAY(1.0, this->tmp);
140
141         // r_{j+1} = sj - omega_j * P * A * sj
142         this->rj.WAXPBY(1.0, this->sj, -omega, this->stmp);
143
144         //-------------------------------------------
145         // One step of BiCGStab iteration ends here
146         //-------------------------------------------
147
148         // Apply several checks for safety
149         if ( numIter >= params.minIter ) {
150             // Compute norm of residual and output iteration information if needed
151             resAbs = rj.Norm2();
152             resRel = resAbs / denAbs;
153             ratio  = resAbs / resAbsOld;
154
155             // Save the best solution so far
156             if ( numIter >= params.safeIter && resAbs < resAbsOld ) safe = x;
157
158             // Apply stagnation checks if it converges slowly
159             if ( ratio > KSM_CHK_RATIO && numIter > params.minIter ) {
160                 // Check I: if solution is close to zero, return ERROR_SOLVER_SOLSTAG
161                 double xNorminf = x.NormInf();
162                 if ( xNorminf < solStagTol ) {
163                     if ( params.verbose > PRINT_MIN )
164                         FASPXX_WARNING("Iteration stopped due to x vanishes!")
165                     errorCode = FaspRetCode::ERROR_SOLVER_SOLSTAG;
166                     break;
167                 }
168
169                 // Check II: if relative difference stagnated, try to restart
170                 double xRelDiff = fabs(alpha) * this->pj.Norm2() / x.Norm2();
171                 if ( (stagStep <= maxStag) && (xRelDiff < solStagTol) ) {
172                     // Compute and update the residual before restart
173                     A->Apply(x, this->rj);
174                     this->rj.XPAY(-1.0, b);
175                     resAbs = this->rj.Norm2();
176                     resRel = resAbs / denAbs;
177                     if ( params.verbose > PRINT_SOME ) {
178                         FASPXX_WARNING("Possible iteration stagnate!")
179                         WarnRealRes(resRel);
180                     }
181
182                     if ( resRel < params.relTol || resAbs < params.absTol ) break;
183                     else {
184                         if ( stagStep >= maxStag ) {
185                             if ( params.verbose > PRINT_MIN )
186                                 FASPXX_WARNING("Iteration stopped due to stagnation!")
187                             errorCode = FaspRetCode::ERROR_SOLVER_STAG;
188                             break;
189                         }
190                         this->pj.SetValues(len, 0.0);
191                         ++stagStep;
192                     }
193
194                     if ( params.verbose > PRINT_SOME ) {
195                         WarnDiffRes(xRelDiff, resRel);
196                         FASPXX_WARNING("Iteration restarted due to stagnation!")
197                     }
198                 } // End of stagnation check!
199             } // End of check I and II
200
201             // Check III: prevent false convergence
202             if ( resRel < params.relTol ) {
203                 // Compute true residual r = b - Ax and update residual
204                 A->Apply(x, this->rj);
205                 this->rj.XPAY(-1.0, b);
206
207                 // Compute residual norms and check convergence
208                 double resRelOld = resRel;
209                 resAbs = rj.Norm2();
210                 resRel = resAbs / denAbs;
211                 if ( resRel < params.relTol || resAbs < params.absTol ) break;
212
213                 if ( params.verbose >= PRINT_MORE ) {
214                     FASPXX_WARNING("False convergence!")
```

```
215                     WarnCompRes(resRelOld);
216                     WarnRealRes(resRel);
217                 }
218 
219                 if ( moreStep >= params.restart ) {
220                     // Note: restart has different meaning here
221                     if ( params.verbose > PRINT_MIN )
222                         FASPXX_WARNING("The tolerance might be too small!")
223                     errorCode = FaspRetCode::ERROR_SOLVER_TOLSMALL;
224                     break;
225                 }
226 
227                 // Prepare for restarting method
228                 this->pj.SetValues(len, 0.0);
229                 ++moreStep;
230             } // End of check!
231         }
232 
233         // Prepare for the next iteration
234         if ( numIter < params.maxIter ) {
235             // Save residual for next iteration
236             resAbsOld = resAbs;
237 
238             // beta_j = (r_{j+1},r0^{*}) / (r_{j},r0^{*}) * alpha_j / omega_j
239             rjr0startmp = rjr0star;
240             rjr0star = this->rj.Dot(this->r0star);
241             beta = rjr0star / rjr0startmp * alpha / omega;
242 
243             // p_{j+1} = r_{j+1} + beta_j * (p_{j} - omega_j * P * A * p_{j})
244             this->tmp.WAXPBY(1.0, this->pj, -omega, this->ptmp);
245             this->pj.WAXPBY(1.0, this->rj, beta, this->tmp);
246         }
247 
248     } // End of main BiCGStab loop
249 
250     // If minIter == numIter == maxIter (preconditioner only), skip this
251     if ( not (numIter == params.minIter && numIter == params.maxIter) ) {
252         this->norm2 = resAbs;
253         this->normInf = rj.NormInf();
254         PrintFinal(numIter, resRel, resAbs, ratio);
255     }
256 
257     // Restore the saved best iteration if needed
258     if ( numIter > params.safeIter ) x = safe;
259 
260     return errorCode;
261 }
```

References SOL::params, and SOLParams::verbose.

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

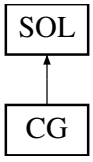
- BiCGStab.hxx
- BiCGStab.cxx

9.2 CG Class Reference

Preconditioned conjugate gradient method.

`#include <CG.hxx>`

Inheritance diagram for CG:



Public Member Functions

- CG ()

  Default constructor.
- ~CG ()=default

  Default destructor.
- FaspRetCode Setup (const LOP &A) override

  Setup the CG method.
- void Clean () override

  Clean up CG data allocated during Setup.
- FaspRetCode Solve (const VEC &b, VEC &x) override

  Solve Ax=b using the CG method.

Additional Inherited Members

9.2.1 Detailed Description

Preconditioned conjugate gradient method.

9.2.2 Member Function Documentation

9.2.2.1 Clean()

```
void CG::Clean ( )  [override], [virtual]
```

Clean up CG data allocated during Setup.

Release additional memory allocated for CG.

Reimplemented from SOL.

```
43 {
44     // Nothing is needed for the moment!
45 }
```

9.2.2.2 Setup()

```
FaspRetCode CG::Setup (
             const LOP & A )  [override], [virtual]
```

Setup the CG method.

Allocate memory, setup coefficient matrix of the linear system.

Reimplemented from SOL.

```
17 {
18     const INT len = A.GetColSize();
19     SetSolType(SOLType::CG); // method type
20
21     // Allocate memory for temporary vectors
22     try {
23         zk.SetValues(len, 0.0);
24         pk.SetValues(len, 0.0);
25         rk.SetValues(len, 0.0);
26         ax.SetValues(len, 0.0);
27         safe.SetValues(len, 0.0);
28     } catch (std::bad_alloc &ex) {
29         return FaspRetCode::ERROR_ALLOC_MEM;
30     }
31
32     // Setup the coefficient matrix
33     this->A = &A;
34
35     // Print used parameters
36     if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);
37
38     return FaspRetCode::SUCCESS;
39 }
```

References SOL::A, CG, ERROR_ALLOC_MEM, LOP::GetColSize(), SOL::params, SOL::SetSolType(), VEC::↩ SetValues(), and SOLParams::verbose.

9.2.2.3 Solve()

```
FaspRetCode CG::Solve (
             const VEC & b,
             VEC & x )  [override], [virtual]
```

Solve Ax=b using the CG method.

Using the Preconditioned Conjugate Gradient method.

Reimplemented from SOL.

```
49 {
50     if ( params.verbose > PRINT_NONE ) std::cout « "Use CG to solve Ax=b ...\n";
51
52     // Check whether vector space sizes match
53     if ( x.GetSize() != A->GetColSize() || b.GetSize() != A->GetRowSize()
54         || A->GetRowSize() != A->GetColSize() )
55         return FaspRetCode::ERROR_NONMATCH_SIZE;
56
57     FaspRetCode errorCode = FaspRetCode::SUCCESS;
58
59     // Local variables
60     const INT len = b.GetSize();
61     const int maxStag = MAX_STAG_NUM; // max number of stagnation checks
62     const double solStagTol = 1e-4 * params.relTol; // solution stagnation tolerance
63     const double solZeroTol = CLOSE_ZERO; // solution close to zero tolerance
64
65     int stagStep = 0, moreStep = 0;
66     double resAbs = 1.0, resRel = 1.0, denAbs = 1.0, ratio = 0.0, resAbsOld = 1.0;
67     double alpha, beta, tmpa, tmpb;
68
69     PrintHead();
```

```
70
71      // Initialize iterative method
72      numIter = 0;
73      A->Apply(x, rk); // A * x -> rk
74      rk.XPAY(-1.0, b); // b - rk -> rk
75
76      // Preconditioned search direction
77      zk.SetValues(len,0.0);
78      pc->Solve(rk, zk); // preconditioning: B(r_k) -> z_k
79
80      // Prepare for the main loop
81      pk = zk;
82      tmpa = zk.Dot(rk);
83
84      // Main CG loop
85      while ( numIter < params.maxIter ) {
86
87          // Start from minIter instead of 0
88          if ( numIter == params.minIter ) {
89              resAbs = rk.Norm2();
90              denAbs = (CLOSE_ZERO > resAbs) ? CLOSE_ZERO : resAbs;
91              resRel = resAbs / denAbs;
92              if (resRel < params.relTol || resAbs < params.absTol) break;
93          }
94
95          if ( numIter >= params.minIter ) PrintInfo(numIter, resRel, resAbs, ratio);
96
97          //-------------------------------------------
98          // CG iteration starts from here
99          //-------------------------------------------
100
101          ++numIter; // iteration count
102
103          A->Apply(pk, ax); // ax = A * p_k, main computational work
104
105          // alpha_k = (z_{k-1}, r_{k-1})/(A*p_{k-1},p_{k-1})
106          tmpb = ax.Dot(pk);
107          if ( fabs(tmpb) > CLOSE_ZERO * CLOSE_ZERO )
108              alpha = tmpa / tmpb;
109          else {
110              FASPXX_WARNING("Divided by zero!")
111              errorCode = FaspRetCode::ERROR_DIVIDE_ZERO;
112              break;
113          }
114
115          // Update solution and residual
116          x.AXPY(alpha, pk);   // x_k = x_{k-1} + alpha_k*p_{k-1}
117          rk.AXPY(-alpha, ax); // r_k = r_{k-1} - alpha_k*A*p_{k-1}
118
119          //-------------------------------------------
120          // One step of CG iteration ends here
121          //-------------------------------------------
122
123          // Apply several checks for robustness
124          if ( numIter >= params.minIter ) {
125              // Compute norm of residual and output iteration information if needed
126              resAbs = rk.Norm2();
127              resRel = resAbs / denAbs;
128              ratio  = resAbs / resAbsOld; // convergence ratio between two steps
129
130              // Save the best solution so far
131              if ( numIter >= params.safeIter && resAbs < resAbsOld ) safe = x;
132
133              // Apply stagnation checks if it converges slowly
134              if ( ratio > KSM_CHK_RATIO ) {
135                  // Check I: if solution is close to zero, return ERROR_SOLVER_SOLSTAG
136                  double xNormInf = x.NormInf();
137                  if (xNormInf < solZeroTol) {
138                      if (params.verbose > PRINT_MIN)
139                          FASPXX_WARNING("Iteration stopped due to x vanishes!")
140                      errorCode = FaspRetCode::ERROR_SOLVER_SOLSTAG;
141                      break;
142                  }
143
144                  // Check II: if relative difference close to zero, try to restart
145                  double xRelDiff = fabs(alpha) * this->pk.Norm2() / x.Norm2();
146                  if ( (stagStep <= maxStag) && (xRelDiff < solStagTol) ) {
147                      // Compute and update the residual before restart
148                      A->Apply(x, this->rk);
149                      this->rk.XPAY(-1.0, b);
150                      resAbs = this->rk.Norm2();
151                      resRel = resAbs / denAbs;
152                      if ( params.verbose > PRINT_SOME ) {
153                          FASPXX_WARNING("Possible iteration stagnate!")
154                          WarnRealRes(resRel);
155                      }
156
```

```
157                     if ( resRel < params.relTol || resAbs < params.absTol ) break;
158                     else {
159                         if ( stagStep >= maxStag ) {
160                             if ( params.verbose > PRINT_MIN )
161                                 FASPXX_WARNING("Iteration stopped due to stagnation!")
162                             errorCode = FaspRetCode::ERROR_SOLVER_STAG;
163                             break;
164                         }
165                         this->pk.SetValues(len, 0.0);
166                         ++stagStep;
167                     }
168 
169                     if ( params.verbose > PRINT_SOME ) {
170                         WarnDiffRes(xRelDiff, resRel);
171                         FASPXX_WARNING("Iteration restarted due to stagnation!")
172                     }
173                 } // End of stagnation check!
174             } // End of check I and II
175 
176             // Check III: prevent false convergence!!!
177             if ( resRel < params.relTol ) {
178                 // Compute and update the true residual r = b - Ax
179                 A->Apply(x, this->rk);
180                 this->rk.XPAY(-1.0, b);
181 
182                 // Compute residual norms and check convergence
183                 double resRelOld = resRel;
184                 resAbs = rk.Norm2();
185                 resRel = resAbs / denAbs;
186                 if ( resRel < params.relTol || resAbs < params.absTol ) break;
187 
188                 // If false converged, print out warning messages
189                 if ( params.verbose >= PRINT_MORE ) {
190                     FASPXX_WARNING("False convergence!")
191                     WarnCompRes(resRelOld);
192                     WarnRealRes(resRel);
193                 }
194 
195                 if ( moreStep >= params.restart ) {
196                     // Note: restart has different meaning here
197                     if ( params.verbose > PRINT_MIN )
198                         FASPXX_WARNING("The tolerance is too small!")
199                     errorCode = FaspRetCode::ERROR_SOLVER_TOLSMALL;
200                     break;
201                 }
202 
203                 // Prepare for restarting method
204                 this->pk.SetValues(0.0);
205                 ++moreStep;
206             } // End of check!
207         }
208 
209         // Prepare for the next iteration
210         if ( numIter < params.maxIter ) {
211             // Save the residual for next iteration
212             resAbsOld = resAbs;
213 
214             // Apply preconditioner z_k = B(r_k)
215             zk.SetValues(len,0.0);
216             pc->Solve(rk, zk);
217 
218             // Compute beta_k = (z_k, r_k) / (z_{k-1}, r_{k-1})
219             tmpb = zk.Dot(rk);
220             beta = tmpb / tmpa;
221             tmpa = tmpb;
222 
223             // Compute p_k = z_k + beta_k*p_{k-1}
224             pk.XPAY(beta, zk);
225         }
226 
227     } // End of main CG loop
228 
229     // If minIter == numIter == maxIter (preconditioner only), skip this
230     if ( not (numIter == params.minIter && numIter == params.maxIter) ) {
231         this->norm2 = resAbs;
232         this->normInf = rk.NormInf();
233         PrintFinal(numIter, resRel, resAbs, ratio);
234     }
235 
236     // Restore the saved best iteration if needed
237     if ( numIter > params.safeIter ) x = safe;
238 
239     return errorCode;
240 }
```

References SOL::params, and SOLParams::verbose.

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

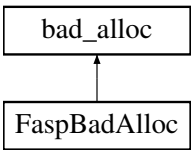
- CG.hxx
- CG.cxx

9.3 FaspBadAlloc Class Reference

Allocation exception capturing class.

`#include <RetCode.hxx>`

Inheritance diagram for FaspBadAlloc:



Public Member Functions

- FaspBadAlloc (const char *file_, const char *func_, const unsigned int line_)

  Default constructor.
- void LogExcep (std::ostream &stream=std::cout) const

  Log allocation error messages in a file or to the screen.

Public Attributes

- const FaspRetCode errorCode = FaspRetCode::SUCCESS

  Error Code.

9.3.1 Detailed Description

Allocation exception capturing class.

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

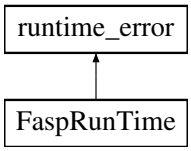
- RetCode.hxx
- RetCode.cxx

9.4 FaspRunTime Class Reference

Run-time exception capturing class.

`#include <RetCode.hxx>`

Inheritance diagram for FaspRunTime:



Public Member Functions

- FaspRunTime (const FaspRetCode code_, const char *file_, const char *func_, const unsigned int line_)
  Default constructor.
- void LogExcep (std::ostream &stream=std::cout) const
  Log exception messages in a file or to the screen.

Public Attributes

- const FaspRetCode errorCode
  Error Code.

9.4.1 Detailed Description

Run-time exception capturing class.

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

- RetCode.hxx
- RetCode.cxx

9.5 GetCycleNum Class Reference

Get CPU-cycle number.

`#include <Timing.hxx>`

Public Member Functions

- __inline__ void Start ()
  Start the cycle count clock.
- __inline__ unsigned long long Stop () const
  Stop the cycle count clock and return number of cycles from start()

9.5.1 Detailed Description

Get CPU-cycle number.

Read the CPU cycles and return number of cycles from start() to stop().

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

- Timing.hxx

9.6 GetWallTime Class Reference

Get elapsed wall-time in millisecond.

`#include <Timing.hxx>`

Public Member Functions

- __inline__ void Start ()
  Start the timer.
- __inline__ double Stop () const
  Stop the timer and return duration from start() in seconds.

9.6.1 Detailed Description

Get elapsed wall-time in millisecond.

Read the current wall-time and return duration from start() to stop().

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

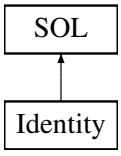
- Timing.hxx

9.7 Identity Class Reference

Identity operator.

`#include <Iter.hxx>`

Inheritance diagram for Identity:



Public Member Functions

- Identity ()
  default constructor
- ~Identity ()
  destructor
- virtual FaspRetCode Solve (const VEC &b, VEC &x)
  Iterator.

Additional Inherited Members

9.7.1 Detailed Description

Identity operator.

9.7.2 Member Function Documentation

9.7.2.1 Solve()

```
FaspRetCode Identity::Solve (
            const VEC & b,
            VEC & x )  [virtual]
```

Iterator.

Does nothing in preconditioning.

Reimplemented from SOL.

```
16 {
17     x = b;
18     return FaspRetCode::SUCCESS;
19 }
```

References SUCCESS.

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

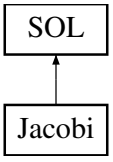
- Iter.hxx
- Iter.cxx

9.8 Jacobi Class Reference

Jacobi iterator.

`#include <Iter.hxx>`

Inheritance diagram for Jacobi:



Public Member Functions

- Jacobi ()

  Default constructor.
- ~Jacobi ()=default

  Default destructor.
- FaspRetCode Setup (const MAT &A)

  Setup the Jacobi method.
- void Clean () override

  Clean up Jacobi data allocated during Setup.
- FaspRetCode Solve (const VEC &b, VEC &x) override

  Solve Ax=b using the Jacobi method.

Public Attributes

- double omega

  Weight for damped or weighted Jacobi.
- VEC diagInv

  Inverse of diagonal entries.
- VEC rk

  Work array for the residual.

Additional Inherited Members

9.8.1 Detailed Description

Jacobi iterator.

9.8.2 Member Function Documentation

9.8.2.1 Setup()

```
FaspRetCode Jacobi::Setup (
              const MAT & A )
```

Setup the Jacobi method.

Setup Jacobi preconditioner.

```
23 {
24     const INT len = A.GetColSize();
25     SetSolType(SOLType::Jacobi); // method type
26 
27     // Allocate memory for temporary vectors
28     try {
29         rk.SetValues(len, 0.0);
30     } catch (std::bad_alloc &ex) {
31         return FaspRetCode::ERROR_ALLOC_MEM;
32     }
33 
34     // Get diagonal and compute its reciprocal
35     A.GetDiag(diagInv);
```

```
36     diagInv.Reciprocal();
37
38     // Setup the coefficient matrix
39     this->A = &A;
40     this->omega = params.weight;
41
42     // Print used parameters if necessary
43     if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);
44
45     return FaspRetCode::SUCCESS;
46 }
```

References SOL::A, diagInv, ERROR_ALLOC_MEM, LOP::GetColSize(), Jacobi, omega, SOL::params, VEC::↩ Reciprocal(), rk, SOL::SetSolType(), VEC::SetValues(), SOLParams::verbose, and SOLParams::weight.

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

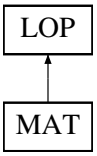
- Iter.hxx
- Iter.cxx

9.9 LOP Class Reference

Linear operator virtual class.

`#include <LOP.hxx>`

Inheritance diagram for LOP:



Public Member Functions

- LOP ()
  Default constructor.
- LOP (const INT &nrow, const INT &mcol)
  Make an LOP from VEC(mcol) to VEC(nrow).
- LOP (const INT &nrow)
  Make an LOP from VEC(nrow) to VEC(nrow).
- LOP (const LOP &lop)
  Make an LOP from another LOP.
- LOP & operator= (const LOP &lop)
  Overload the = operator.
- ∼LOP ()=default
  Default destructor.
- INT GetRowSize () const
  Get row space dimension.
- INT GetColSize () const
  Get column space dimension.
- virtual void Apply (const VEC &x, VEC &y) const
  Action of the linear operator to a vector.

Protected Attributes

- INT nrow

  number of rows
- INT mcol

  number of columns

9.9.1 Detailed Description

Linear operator virtual class.

9.9.2 Constructor & Destructor Documentation

9.9.2.1 LOP() [1/3]

```
LOP::LOP (
              const INT & nrow,
              const INT & mcol )
```

Make an LOP from VEC(mcol) to VEC(nrow).

Assign nrow, mcol to ∗this.

```
16 {
17     this->nrow = nrow;
18     this->mcol = mcol;
19 }
```

References mcol, and nrow.

9.9.2.2 LOP() [2/3]

```
LOP::LOP (
              const INT & nrow )  [explicit]
```

Make an LOP from VEC(nrow) to VEC(nrow).

Assign nrow, mcol=nrow to ∗this.

```
23 {
24     this->nrow = nrow;
25     this->mcol = nrow;
26 }
```

References mcol, and nrow.

9.9.2.3 LOP() [3/3]

```
LOP::LOP (
            const LOP & lop )
```

Make an LOP from another LOP.

Assign LOP object to ∗this.

```
30 {
31     this->nrow = lop.nrow;
32     this->mcol = lop.mcol;
33 }
```

References mcol, and nrow.

9.9.3 Member Function Documentation

9.9.3.1 GetColSize()

```
INT LOP::GetColSize ( ) const
```

Get column space dimension.

Dimension of the column space of LOP.

```
51 {
52     return this->mcol;
53 }
```

References mcol.

9.9.3.2 GetRowSize()

```
INT LOP::GetRowSize ( ) const
```

Get row space dimension.

Dimension of the row space of LOP.

```
45 {
46     return this->nrow;
47 }
```

References nrow.

9.9.3.3 operator=()

```
LOP & LOP::operator= (
            const LOP & lop )
```

Overload the = operator.

Assignment for the LOP object.

```
37 {
38     this->nrow = lop.nrow;
39     this->mcol = lop.mcol;
40     return *this;
41 }
```

References mcol, and nrow.

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

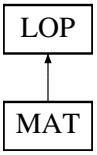
- LOP.hxx
- LOP.cxx

9.10 MAT Class Reference

Sparse matrix class.

`#include <MAT.hxx>`

Inheritance diagram for MAT:



Public Member Functions

- MAT ()

  Default constructor.
- MAT (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, const std::vector< INT > &diagPtr)

  Construct sparse matrix from a CSRx matrix.
- MAT (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr)

  Construct sparse matrix from a CSR matrix.
- MAT (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr)

  Construct sparsity structure from a CSR matrix.
- MAT (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, const std::vector< INT > &diagPtr)

  Construct sparsity structure from a CSRx matrix.
- MAT (const VEC &v)

Construct diagonal MAT matrix from a VEC object.

- MAT (const std::vector< DBL > &v)

  Construct diagonal MAT matrix from a vector object.

- MAT (const MAT &mat)

  Clone from another MAT.

- ~MAT ()=default

  Default destructor.

- MAT & operator= (const MAT &mat)

  Overload = operator.

- void SetValues (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, const std::vector< INT > &diagPtr)

  Set values of the matrix with CSRx format.

- void SetValues (const INT &nrow, const INT &mcol, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr)

  Set values of the matrix with CSR format.

- INT GetNNZ () const

  Get number of nonzeros of the matrix.

- void GetDiag (VEC &v) const

  Get the diagonal entries of ∗this and save them in a VEC object.

- void GetDiagInv (MAT &m) const

  Get reciprocal diagonal entries and save them in a MAT object.

- void GetLowerTri (MAT &lTri) const

  Get the lower triangular matrix.

- void GetUpperTri (MAT &uTri) const

  Get the upper triangular matrix.

- void CopyTo (MAT &mat) const

  Copy the matrix to another MAT object.

- void Scale (const DBL a)

  Scale the matrix with a scalar.

- void Shift (const DBL a)

  Shift the matrix with a scalar matrix.

- void Zero ()

  Set the matrix to a zero matrix.

- void Apply (const VEC &v, VEC &w) const

  Sparse matrix-vector multiplication.

- void Transpose ()

  Transpose of the matrix.

- void MultTransposeAdd (const VEC &v1, const VEC &v2, VEC &v) const

  Compute transpose of A multiply by v1 plus v2.

- DBL GetValue (const INT &row, const INT &col) const

  Get the value of [i,j]-entry of the matrix.

- void Add (const DBL a, const MAT &mat1, const DBL b, const MAT &mat2)

  ∗this = a ∗ mat1 + b ∗ mat2

- void Mult (const MAT &matl, const MAT &matr)

  ∗this = matl ∗ matr

- void MultLeft (const MAT &mat)

  ∗this = ∗this ∗ mat

- void MultRight (const MAT &mat)

  ∗this = mat ∗ ∗this

Friends

- void WriteCSR (char *filename, MAT mat)

  Write an MAT matrix to a disk file in CSR format.
- void WriteMTX (char *filename, MAT mat)

  Write an MAT matrix to a disk file in MTX format.

Additional Inherited Members

9.10.1 Detailed Description

Sparse matrix class.

9.10.2 Constructor & Destructor Documentation

9.10.2.1 MAT() [1/7]

```
MAT::MAT (
              const INT & nrow,
              const INT & mcol,
              const INT & nnz,
              const std::vector< DBL > & values,
              const std::vector< INT > & colInd,
              const std::vector< INT > & rowPtr,
              const std::vector< INT > & diagPtr )
```

Construct sparse matrix from a CSRx matrix.

Assign nrow, mcol, nnz, values, colInd, rowPtr, diagPtr to *this.

```
19                                                                    {
20     if (nrow == 0 || mcol == 0 || nnz == 0) {
21         this->Empty();
22         return;
23     }
24
25     this->nrow = nrow;
26     this->mcol = mcol;
27     this->nnz = nnz;
28     this->values = values;
29     this->colInd = colInd;
30     this->rowPtr = rowPtr;
31     this->diagPtr = diagPtr;
32 }
```

References LOP::mcol, and LOP::nrow.

9.10.2.2 MAT() [2/7]

```
MAT::MAT (
              const INT & nrow,
              const INT & mcol,
              const INT & nnz,
              const std::vector< DBL > & values,
              const std::vector< INT > & colInd,
              const std::vector< INT > & rowPtr )
```

Construct sparse matrix from a CSR matrix.

Assign nrow, mcol, nnz, values, colInd, rowPtr to ∗this and generate diagPtr.

```
37                                                {
38     if (nrow == 0 || mcol == 0 || nnz == 0) {
39         this->Empty();
40         return;
41     }
42
43     this->nrow = nrow;
44     this->mcol = mcol;
45     this->nnz = nnz;
46     this->values = values;
47     this->colInd = colInd;
48     this->rowPtr = rowPtr;
49     this->FormDiagPtr();
50 }
```

References LOP::mcol, and LOP::nrow.

9.10.2.3 MAT() [3/7]

```
MAT::MAT (
              const INT & nrow,
              const INT & mcol,
              const INT & nnz,
              const std::vector< INT > & colInd,
              const std::vector< INT > & rowPtr )
```

Construct sparsity structure from a CSR matrix.

Assign nrow, mcol, nnz, colInd, rowPtr to ∗this and generate diagPtr.

```
54                                                                                {
55     if (nrow == 0 || mcol == 0 || nnz == 0) {
56         this->Empty();
57         return;
58     }
59
60     this->nrow = nrow;
61     this->mcol = mcol;
62     this->nnz = nnz;
63     this->colInd = colInd;
64     this->rowPtr = rowPtr;
65     this->values.resize(0);
66     this->FormDiagPtr();
67 }
```

References LOP::mcol, and LOP::nrow.

9.10.2.4 MAT() [4/7]

```
MAT::MAT (
            const INT & nrow,
            const INT & mcol,
            const INT & nnz,
            const std::vector< INT > & colInd,
            const std::vector< INT > & rowPtr,
            const std::vector< INT > & diagPtr )
```

Construct sparsity structure from a CSRx matrix.

Assign nrow, mcol, nnz, colInd, rowPtr, diagPtr to ∗this.

```
72                                          {
73     if (nrow == 0 || mcol == 0 || nnz == 0) {
74         this->Empty();
75         return;
76     }
77
78     this->nrow = nrow;
79     this->mcol = mcol;
80     this->nnz = nnz;
81     this->colInd = colInd;
82     this->rowPtr = rowPtr;
83     this->diagPtr = diagPtr;
84     this->values.resize(0);
85 }
```

References LOP::mcol, and LOP::nrow.

9.10.2.5 MAT() [5/7]

```
MAT::MAT (
            const VEC & v )  [explicit]
```

Construct diagonal MAT matrix from a VEC object.

Assign diagonal values from a VEC to ∗this.

```
88                      {
89     INT size = v.GetSize();
90
91     // Return an empty matrix if size==0
92     if (size == 0) {
93         this->Empty();
94         return;
95     }
96
97     // Set MAT size
98     this->nrow = size;
99     this->mcol = size;
100     this->nnz = size;
101
102     INT *p;
103     try {
104         p = new INT[size + 1];
105     } catch (std::bad_alloc &ex) {
106         this->nrow = 0;
107         this->mcol = 0;
108         this->nnz = 0;
109         throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
110     }
111
112     // Set values from v
113     this->values.resize(size);
114     for (INT j = 0; j < size; ++j) this->values[j] = v.values[j];
115
116     // Set colInd to {0, 1, ..., size-1}
117     for (INT j = 0; j <= size; ++j) p[j] = j;
118     this->colInd.resize(size);
119     this->colInd.assign(p, p + size);
```

```
120
121     // Set rowPtr to {0, 1, ..., size}
122     this->rowPtr.resize(size + 1);
123     this->rowPtr.assign(p, p + size + 1);
124
125     // Set diagPtr to {0, 1, ..., size-1}
126     this->diagPtr.resize(size);
127     this->diagPtr.assign(p, p + size);
128
129     delete[] p;
130 }
```

References VEC::GetSize(), LOP::mcol, and LOP::nrow.

9.10.2.6 MAT() [6/7]

```
MAT::MAT (
            const std::vector< DBL > & v )  [explicit]
```

Construct diagonal MAT matrix from a vector object.

Assign diagonal values from a vector to ∗this.

```
133                                 {
134     const INT size = vt.size();
135
136     // Return an empty matrix if size==0
137     if (size == 0) {
138         this->Empty();
139         return;
140     }
141
142     // Set MAT size
143     this->nrow = size;
144     this->mcol = size;
145     this->nnz = size;
146
147     INT *p;
148     try {
149         p = new INT[size + 1];
150     } catch (std::bad_alloc &ex) {
151         this->nrow = 0;
152         this->mcol = 0;
153         this->nnz = 0;
154         throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
155     }
156
157     // Set values from vt
158     this->values.resize(size);
159     this->values.assign(vt.begin(), vt.begin() + size);
160
161     // Set colInd to {0, 1, ..., size-1}
162     for (INT j = 0; j <= size; ++j) p[j] = j;
163     this->colInd.resize(size);
164     this->colInd.assign(p, p + size);
165
166     // Set rowPtr to {0, 1, ..., size}
167     this->rowPtr.resize(size + 1);
168     this->rowPtr.assign(p, p + size + 1);
169
170     // Set diagPtr to {0, 1, ..., size-1}
171     this->diagPtr.resize(size);
172     this->diagPtr.assign(p, p + size);
173
174     delete[] p;
175 }
```

References LOP::mcol, and LOP::nrow.

9.10.2.7 MAT() [7/7]

```
MAT::MAT (
            const MAT & mat )
```

Clone from another MAT.

Assign MAT object to ∗this.

```
178                         {
179     this->nrow = mat.nrow;
180     this->mcol = mat.mcol;
181     this->nnz = mat.nnz;
182     this->values = mat.values;
183     this->colInd = mat.colInd;
184     this->rowPtr = mat.rowPtr;
185     this->diagPtr = mat.diagPtr;
186 }
```

References LOP::mcol, and LOP::nrow.

9.10.3 Member Function Documentation

9.10.3.1 Add()

```
void MAT::Add (
            const DBL a,
            const MAT & mat1,
            const DBL b,
            const MAT & mat2 )
```

∗this = a ∗ mat1 + b ∗ mat2

∗this = a ∗ mat1 + b ∗ mat2.

```
588                                                                 {
589
590     MAT tmpMat;
591     INT i, j, k, l;
592     INT count = 0, added, countrow;
593
594     if (mat1.nnz == 0) {
595         tmpMat = mat2;
596         tmpMat.Scale(b);
597         return;
598     }
599
600     if (mat2.nnz == 0) {
601         tmpMat = mat1;
602         tmpMat.Scale(a);
603         return;
604     }
605
606     tmpMat.nrow = mat1.nrow;
607     tmpMat.mcol = mat1.mcol;
608
609     tmpMat.rowPtr.resize(tmpMat.nrow + 1);
610     tmpMat.colInd.resize(mat1.nnz + mat2.nnz);
611     tmpMat.values.resize(mat1.nnz + mat2.nnz);
612
613     tmpMat.colInd.assign(mat1.nnz + mat2.nnz, -1);
614
615     for (i = 0; i < mat1.nrow; ++i) {
616         countrow = 0;
617         for (j = mat1.rowPtr[i]; j < mat1.rowPtr[i + 1]; ++j) {
618             tmpMat.values[count] = a * mat1.values[j];
619             tmpMat.colInd[count] = mat1.colInd[j];
620             ++tmpMat.rowPtr[i + 1];
621             ++count;
```

```
622             ++countrow;
623         }
624 
625         for (k = mat2.rowPtr[i]; k < mat2.rowPtr[i + 1]; ++k) {
626             added = 0;
627             for (l = tmpMat.rowPtr[i]; l < tmpMat.rowPtr[i] + countrow + 1; ++l) {
628                 if (mat2.colInd[k] == tmpMat.colInd[l]) {
629                     tmpMat.values[l] = tmpMat.values[l] + b * mat2.values[k];
630                     added = 1;
631                     break;
632                 }
633             }
634             if (added == 0) {
635                 tmpMat.values[count] = b * mat2.values[k];
636                 tmpMat.colInd[count] = mat2.colInd[k];
637                 ++tmpMat.rowPtr[i + 1];
638                 ++count;
639             }
640         }
641         tmpMat.rowPtr[i + 1] += tmpMat.rowPtr[i];
642     }
643     tmpMat.nnz = count;
644     tmpMat.colInd.resize(count);
645     tmpMat.values.resize(count);
646     tmpMat.colInd.shrink_to_fit();
647     tmpMat.values.shrink_to_fit();
648 
649     SortCSRRow(tmpMat.nrow, tmpMat.mcol, tmpMat.nnz, tmpMat.rowPtr, tmpMat.colInd,
650                tmpMat.values);
651 
652     tmpMat.FormDiagPtr();
653     *this = tmpMat;
654 }
```

References LOP::mcol, LOP::nrow, Scale(), and SortCSRRow().

9.10.3.2 Apply()

```
void MAT::Apply (
             const VEC & v,
             VEC & w ) const  [virtual]
```

Sparse matrix-vector multiplication.

Compute w = ∗this ∗ v.

Reimplemented from LOP.

```
358                                         {
359     INT begin, i, k;
360 
361     if ( !this->values.empty() ) { // Regular sparse matrix
362         for ( i = 0; i < this->nrow; ++i ) {
363             begin = this->rowPtr[i];
364             switch (this->rowPtr[i + 1] - begin) {
365                 case 4:
366                     w.values[i] = this->values[begin]
367                                   * v.values[this->colInd[begin]];
368                     w.values[i] += this->values[begin + 1]
369                                    * v.values[this->colInd[begin + 1]];
370                     w.values[i] += this->values[begin + 2]
371                                    * v.values[this->colInd[begin + 2]];
372                     w.values[i] += this->values[begin + 3]
373                                    * v.values[this->colInd[begin + 3]];
374                     break;
375                 case 5:
376                     w.values[i] = this->values[begin]
377                                   * v.values[this->colInd[begin]];
378                     w.values[i] += this->values[begin + 1]
379                                    * v.values[this->colInd[begin + 1]];
380                     w.values[i] += this->values[begin + 2]
381                                    * v.values[this->colInd[begin + 2]];
382                     w.values[i] += this->values[begin + 3]
383                                    * v.values[this->colInd[begin + 3]];
384                     w.values[i] += this->values[begin + 4]
385                                    * v.values[this->colInd[begin + 4]];
```

```
386                     break;
387                 case 6:
388                     w.values[i] = this->values[begin]
389                                   * v.values[this->colInd[begin]];
390                     w.values[i] += this->values[begin + 1]
391                                    * v.values[this->colInd[begin + 1]];
392                     w.values[i] += this->values[begin + 2]
393                                    * v.values[this->colInd[begin + 2]];
394                     w.values[i] += this->values[begin + 3]
395                                    * v.values[this->colInd[begin + 3]];
396                     w.values[i] += this->values[begin + 4]
397                                    * v.values[this->colInd[begin + 4]];
398                     w.values[i] += this->values[begin + 5]
399                                    * v.values[this->colInd[begin + 5]];
400                     break;
401                 default:
402                     w.values[i] =
403                             this->values[begin] * v.values[this->colInd[begin]];
404                     for (k = begin + 1; k < this->rowPtr[i + 1]; ++k)
405                         w.values[i] += this->values[k] * v.values[this->colInd[k]];
406             }
407         }
408     } else { // Only sparse structure
409         for ( i = 0; i < this->nrow; ++i ) {
410             begin = this->rowPtr[i];
411             switch (this->rowPtr[i + 1] - begin) {
412                 case 4:
413                     w.values[i] = v.values[this->colInd[begin]];
414                     w.values[i] += v.values[this->colInd[begin + 1]];
415                     w.values[i] += v.values[this->colInd[begin + 2]];
416                     w.values[i] += v.values[this->colInd[begin + 3]];
417                     break;
418                 case 5:
419                     w.values[i] = v.values[this->colInd[begin]];
420                     w.values[i] += v.values[this->colInd[begin + 1]];
421                     w.values[i] += v.values[this->colInd[begin + 2]];
422                     w.values[i] += v.values[this->colInd[begin + 3]];
423                     w.values[i] += v.values[this->colInd[begin + 4]];
424                     break;
425                 case 6:
426                     w.values[i] = v.values[this->colInd[begin]];
427                     w.values[i] += v.values[this->colInd[begin + 1]];
428                     w.values[i] += v.values[this->colInd[begin + 2]];
429                     w.values[i] += v.values[this->colInd[begin + 3]];
430                     w.values[i] += v.values[this->colInd[begin + 4]];
431                     w.values[i] += v.values[this->colInd[begin + 5]];
432                     break;
433                 default:
434                     w.values[i] = v.values[this->colInd[begin]];
435                     for (k = begin + 1; k < this->rowPtr[i + 1]; ++k)
436                         w.values[i] += v.values[this->colInd[k]];
437             }
438         }
439     } // end if values.size > 0
440 }
```

References LOP::nrow.

9.10.3.3 CopyTo()

```
void MAT::CopyTo (
             MAT & mat ) const
```

Copy the matrix to another MAT object.

Copy *this to mat.

```
334                                 {
335     mat = *this;
336 }
```

9.10.3.4 GetDiagInv()

```
void MAT::GetDiagInv (
            MAT & m ) const
```

Get reciprocal diagonal entries and save them in a MAT object.

Get the diagonal entries' reciprocal of ∗this and save them in a MAT object.

```
256                                 {
257     m.nrow = this->nrow;
258     m.mcol = this->mcol;
259     m.nnz = this->nrow;
260
261     m.rowPtr.resize(m.nrow + 1);
262     for ( INT j = 0; j < m.nrow + 1; ++j ) m.rowPtr[j] = j;
263
264     m.diagPtr.resize(m.nrow);
265     for ( INT j = 0; j < m.nrow; ++j ) m.diagPtr[j] = j;
266
267     m.colInd.resize(m.nnz);
268     for ( INT j = 0; j < m.nnz; ++j ) m.colInd[j] = j;
269
270     m.values.resize(m.nnz);
271     for ( INT j = 0; j < m.nnz; ++j )
272         m.values[j] = 1.0 / this->values[this->diagPtr[j]];
273 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.5 GetNNZ()

```
INT MAT::GetNNZ ( ) const
```

Get number of nonzeros of the matrix.

Return this->nnz.

```
238                         {
239     return this->nnz;
240 }
```

9.10.3.6 GetValue()

```
DBL MAT::GetValue (
             const INT & irow,
             const INT & jcol ) const
```

Get the value of [i,j]-entry of the matrix.

Get (∗this)[i][j].

Note

If ∗this is a sparse structure, it will return 1.0 for nonzero entries.

```
572                                                         {
573     if ( this->colInd[this->rowPtr[irow]] <= jcol &&
574          this->colInd[this->rowPtr[irow + 1] - 1] >= jcol ) {
575         for ( INT j = this->rowPtr[irow]; j < this->rowPtr[irow + 1]; ++j ) {
576             if ( jcol == this->colInd[j] ) {
577                 if ( this->values.empty() )
578                     return 1.0; // sparse structure indicator
579                 else
580                     return this->values[j];
581             }
582         }
583     }
584     return 0.0;
585 }
```

9.10.3.7 Mult()

```
void MAT::Mult (
            const MAT & matl,
            const MAT & matr )
```

∗this = matl ∗ matr

∗this = matl ∗ matr.

```
657                                             {
658     INT l, count;
659     INT *tmp = new INT[matr.mcol];
660
661     MAT mat;
662
663     this->nrow = matl.nrow;
664     this->mcol = matr.mcol;
665     this->rowPtr.resize(this->nrow + 1);
666
667     for (INT i = 0; i < matr.mcol; ++i) tmp[i] = -1;
668
669     for (INT i = 0; i < this->nrow; ++i) {
670         count = 0;
671         for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
672             for (INT j = matr.rowPtr[matl.colInd[k]];
673                  j < matr.rowPtr[matl.colInd[k] + 1]; ++j) {
674                 for (l = 0; l < count; ++l) {
675                     if (tmp[l] == matr.colInd[j]) break;
676                 }
677                 if (l == count) {
678                     tmp[count] = matr.colInd[j];
679                     ++count;
680                 }
681             }
682         }
683         this->rowPtr[i + 1] = count;
684         for (INT j = 0; j < count; ++j) tmp[j] = -1;
685     }
686
687     for (INT i = 0; i < this->nrow; ++i) this->rowPtr[i + 1] += this->rowPtr[i];
688
689     INT count_tmp;
690
691     this->colInd.resize(this->rowPtr[this->nrow]);
692
693     for (INT i = 0; i < this->nrow; ++i) {
694         count_tmp = 0;
695         count = this->rowPtr[i];
696         for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
697             for (INT j = matr.rowPtr[matl.colInd[k]];
698                  j < matr.rowPtr[matl.colInd[k] + 1]; ++j) {
699                 for (l = 0; l < count_tmp; ++l) {
700                     if (tmp[l] == matr.colInd[j]) break;
701                 }
702                 if (l == count_tmp) {
703                     this->colInd[count] = matr.colInd[j];
704                     tmp[count_tmp] = matr.colInd[j];
705                     ++count;
706                     ++count_tmp;
707                 }
708             }
709         }
710
711         for (INT j = 0; j < count_tmp; ++j) tmp[j] = -1;
712     }
713
714     delete[] tmp;
715
716     this->values.resize(this->rowPtr[this->nrow]);
717
718     for (INT i = 0; i < this->nrow; ++i) {
719         for (INT j = this->rowPtr[i]; j < this->rowPtr[i + 1]; ++j) {
720             this->values[j] = 0;
721             for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
722                 for (l = matr.rowPtr[matl.colInd[k]];
723                      l < matr.rowPtr[matl.colInd[k] + 1]; ++l) {
724                     if (matr.colInd[l] == this->colInd[j])
725                         this->values[j] += matl.values[k] * matr.values[l];
726                 }
727             }
728         }
729     }
```

```
730
731     this->nnz = this->rowPtr[this->nrow] - this->rowPtr[0];
732
733     this->FormDiagPtr();
734 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.8 MultLeft()

```
void MAT::MultLeft (
             const MAT & mat )
```

∗this = ∗this ∗ mat

∗this = ∗this ∗ mat.

```
737                                  {
738     MAT tmp;
739     tmp = *this;
740
741     Mult(tmp, mat);
742 }
```

References Mult().

9.10.3.9 MultRight()

```
void MAT::MultRight (
             const MAT & mat )
```

∗this = mat ∗ ∗this

∗this = mat ∗ ∗this.

```
745                                   {
746     MAT tmp;
747     tmp = *this;
748
749     Mult(mat, tmp);
750 }
```

References Mult().

9.10.3.10 MultTransposeAdd()

```
void MAT::MultTransposeAdd (
            const VEC & v1,
            const VEC & v2,
            VEC & v ) const
```

Compute transpose of A multiply by v1 plus v2.

Compute v = A'∗v1 + v2.

```
504                                                                     {
505     const INT n = this->nrow, m = this->mcol, nnz = this->nnz;
506     INT i, j, k, p;
507
508     MAT tmp;
509     tmp.nrow = m;
510     tmp.mcol = n;
511     tmp.nnz = nnz;
512
513     try {
514         tmp.rowPtr.resize(m + 1);
515         tmp.colInd.resize(nnz);
516     } catch (std::bad_alloc &ex) {
517         throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
518     }
519
520     if ( !this->values.empty() ) {
521         try {
522             tmp.values.resize(nnz);
523         } catch (std::bad_alloc &ex) {
524             throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
525         }
526     } else {
527         tmp.values.resize(0);
528     }
529
530     for ( j = 0; j < nnz; ++j ) {
531         i = this->colInd[j];
532         if ( i < m - 1 ) ++tmp.rowPtr[i + 2];
533     }
534
535     for ( i = 2; i <= m; ++i ) tmp.rowPtr[i] += tmp.rowPtr[i - 1];
536
537     if ( !this->values.empty() ) {
538         for (i = 0; i < n; ++i) {
539             INT begin = this->rowPtr[i];
540             for (p = begin; p < this->rowPtr[i + 1]; ++p) {
541                 j = this->colInd[p] + 1;
542                 k = tmp.rowPtr[j];
543                 tmp.colInd[k] = i;
544                 tmp.values[k] = this->values[p];
545                 tmp.rowPtr[j] = k + 1;
546             }
547         }
548     } else {
549         for (i = 0; i < n; ++i) {
550             INT begin = this->rowPtr[i];
551             for (p = begin; p < this->rowPtr[i + 1]; ++p) {
552                 j = this->colInd[p] + 1;
553                 k = tmp.rowPtr[j];
554                 tmp.colInd[k] = i;
555                 tmp.rowPtr[j] = k + 1;
556             }
557         }
558     }
559
560     v = v2;
561
562     INT begin;
563     for (i = 0; i < tmp.nrow; ++i) {
564         begin = tmp.rowPtr[i];
565         for (j = begin; j < this->rowPtr[i + 1]; ++j)
566             v.values[i] += v1.values[tmp.colInd[j]] * tmp.values[j];
567     }
568 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.11 operator=()

```
MAT & MAT::operator= (
            const MAT & mat )
```

Overload = operator.

Assignment for the MAT object.

```
189                                     {
190     this->nrow = mat.nrow;
191     this->mcol = mat.mcol;
192     this->nnz = mat.nnz;
193     this->values = mat.values;
194     this->colInd = mat.colInd;
195     this->rowPtr = mat.rowPtr;
196     this->diagPtr = mat.diagPtr;
197     return *this;
198 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.12 Scale()

```
void MAT::Scale (
            const DBL a )
```

Scale the matrix with a scalar.

Scale *this *= a.

```
339                             {
340     if ( this->values.empty() ) return; // MAT is a sparse structure!!!
341
342     for ( INT j = 0; j < this->nnz; ++j ) this->values[j] *= a;
343 }
```

9.10.3.13 SetValues() [1/2]

```
void MAT::SetValues (
            const INT & nrow,
            const INT & mcol,
            const INT & nnz,
            const std::vector< DBL > & values,
            const std::vector< INT > & colInd,
            const std::vector< INT > & rowPtr )
```

Set values of the matrix with CSR format.

Set values of nrow, mcol, nnz, values, rowPtr, colInd.

```
222                                                     {
223     if (nrow == 0 || mcol == 0 || nnz == 0) {
224         this->Empty();
225         return;
226     }
227
228     this->nrow = nrow;
229     this->mcol = mcol;
230     this->nnz = nnz;
231     this->rowPtr = rowPtr;
232     this->colInd = colInd;
233     this->values = values;
234     this->FormDiagPtr();
235 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.14 SetValues() [2/2]

```
void MAT::SetValues (
            const INT & nrow,
            const INT & mcol,
            const INT & nnz,
            const std::vector< DBL > & values,
            const std::vector< INT > & colInd,
            const std::vector< INT > & rowPtr,
            const std::vector< INT > & diagPtr )
```

Set values of the matrix with CSRx format.

Set values of nrow, mcol, nnz, values, colInd, rowPtr, diagPtr.

```
204                                                         {
205     if (nrow == 0 || mcol == 0 || nnz == 0) {
206         this->Empty();
207         return;
208     }
209
210     this->nrow = nrow;
211     this->mcol = mcol;
212     this->nnz = nnz;
213     this->values = values;
214     this->rowPtr = rowPtr;
215     this->colInd = colInd;
216     this->diagPtr = diagPtr;
217 }
```

References LOP::mcol, and LOP::nrow.

9.10.3.15 Shift()

```
void MAT::Shift (
            const DBL a )
```

Shift the matrix with a scalar matrix.

Shift ∗this += a ∗ I.

```
346                             {
347     if ( this->values.empty() ) return; // MAT is a sparse structure!!!
348
349     for ( INT j : this->diagPtr ) this->values[j] += a;
350 }
```

9.10.3.16 Transpose()

```
void MAT::Transpose ( )
```

Transpose of the matrix.

Transpose ∗this in place.

```
443                     {
444     const INT n = this->nrow, m = this->mcol, nnz = this->nnz;
445     INT i, j, k, p;
446
447     MAT tmp;
448     tmp.nrow = this->mcol;
449     tmp.mcol = this->nrow;
450     tmp.nnz = this->nnz;
```

```
451
452     try {
453         tmp.rowPtr.resize(this->mcol + 1);
454         tmp.colInd.resize(nnz);
455     } catch (std::bad_alloc &ex) {
456         throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
457     }
458
459     if (!this->values.empty()) {
460         try {
461             tmp.values.resize(nnz);
462         } catch (std::bad_alloc &ex) {
463             throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
464         }
465     } else {
466         tmp.values.resize(0);
467     }
468
469     for ( INT j = 0; j < nnz; ++j ) {
470         i = this->colInd[j];
471         if ( i < m - 1 ) ++tmp.rowPtr[i + 2];
472     }
473
474     for ( i = 2; i <= m; ++i ) tmp.rowPtr[i] += tmp.rowPtr[i - 1];
475
476     if ( !this->values.empty() ) {
477         for ( i = 0; i < n; ++i ) {
478             INT begin = this->rowPtr[i];
479             for ( p = begin; p < this->rowPtr[i + 1]; ++p ) {
480                 j = this->colInd[p] + 1;
481                 k = tmp.rowPtr[j];
482                 tmp.colInd[k] = i;
483                 tmp.values[k] = this->values[p];
484                 tmp.rowPtr[j] = k + 1;
485             }
486         }
487     } else {
488         for ( i = 0; i < n; ++i ) {
489             INT begin = this->rowPtr[i];
490             for ( p = begin; p < this->rowPtr[i + 1]; ++p ) {
491                 j = this->colInd[p] + 1;
492                 k = tmp.rowPtr[j];
493                 tmp.colInd[k] = i;
494                 tmp.rowPtr[j] = k + 1;
495             }
496         }
497     }
498
499     tmp.FormDiagPtr();
500     this->operator=(tmp);
501 }
```

References LOP::mcol, LOP::nrow, and operator=().

9.10.3.17 Zero()

```
void MAT::Zero ( )
```

Set the matrix to a zero matrix.

Set all the entries to zero, without changing matrix size.

```
353                 {
354     for (INT j = 0; j < this->nnz; ++j) values[j] = 0.0;
355 }
```

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

- MAT.hxx
- MAT.cxx

9.11 Parameters Class Reference

Solver parameters.

`#include <Param.hxx>`

Public Member Functions

- Parameters (int _argc, const char *_argv[ ])
  Default constructor.
- ~Parameters ()=default
  Default destructor.
- void AddParam (const std::string &name, const std::string &help, bool *ptr, int marker=0)
  Add a bool type parameter.
- void AddParam (const std::string &name, const std::string &help, int *ptr, int marker=0)
  Add an int type parameter.
- void AddParam (const std::string &name, const std::string &help, double *ptr, int marker=0)
  Add a double type parameter.
- void AddParam (const std::string &name, const std::string &help, std::string *ptr, int marker=0)
  Add a string type parameter.
- void AddParam (const std::string &name, const std::string &help, Output *ptr, int marker=0)
  Add a Output type parameter.
- void Parse ()
  Parse the parameters.
- void PrintUserParams (std::ostream &out) const
  Print original params (before merge or parse) in user program.
- void PrintFileParams (std::ostream &out) const
  Print parameters coming from an option file.
- void PrintCommandLineParams (std::ostream &out) const
  Print parameters coming from command line input.
- void Print (std::ostream &out=std::cout) const
  Print parameters used in user code (after merge or parse).
- void PrintHelp (std::ostream &out=std::cout) const
  Print the help messages for Param.

9.11.1 Detailed Description

Solver parameters.

9.11.2 Member Function Documentation

9.11.2.1 AddParam() [1/5]

```
void Parameters::AddParam (
            const std::string & name,
            const std::string & help,
            bool * ptr,
            int marker = 0 )
```

Add a bool type parameter.

Bool type parameter.

```
182 {
183     paramsUser.emplace_back(BoolType, name, help, ptr, marker);
184 }
```

9.11.2.2 AddParam() [2/5]

```
void Parameters::AddParam (
            const std::string & name,
            const std::string & help,
            double * ptr,
            int marker = 0 )
```

Add a double type parameter.

Double type parameter.

```
194 {
195     paramsUser.emplace_back(DoubleType, name, help, ptr, marker);
196 }
```

9.11.2.3 AddParam() [3/5]

```
void Parameters::AddParam (
            const std::string & name,
            const std::string & help,
            int * ptr,
            int marker = 0 )
```

Add an int type parameter.

Int type parameter.

```
188 {
189     paramsUser.emplace_back(IntType, name, help, ptr, marker);
190 }
```

9.11.2.4 AddParam() [4/5]

```
void Parameters::AddParam (
            const std::string & name,
            const std::string & help,
            Output * ptr,
            int marker = 0 )
```

Add a Output type parameter.

Output type parameter.

```
206 {
207     paramsUser.emplace_back(OutputType, name, help, ptr, marker);
208 }
```

9.11.2.5 AddParam() [5/5]

```
void Parameters::AddParam (
            const std::string & name,
            const std::string & help,
            std::string * ptr,
            int marker = 0 )
```

Add a string type parameter.

String type parameter.

```
200 {
201     paramsUser.emplace_back(StringType, name, help, ptr, marker);
202 }
```

9.11.2.6 Parse()

```
void Parameters::Parse ( )
```

Parse the parameters.

Main entrance point for reading and handling parameters.

```
212 {
213     // Read parameters
214     ReadFromCommandLine();
215     ReadFromFile();
216 
217     // Save the original and then merge
218     SaveUserParams(paramsUserOrg);
219     MergeParams();
220 }
```

9.11.2.7 Print()

```
void Parameters::Print (
            std::ostream & out = std::cout ) const
```

Print parameters used in user code (after merge or parse).

Print parameters used with width adapt to the names.

```
268 {
269     size_t max_len = 0;
270     for ( const auto& itm: paramsUser ) {
271         if ( itm.paramName.length() > max_len ) max_len = itm.paramName.length();
272     }
273 
274     static std::string indent = "    ";
275     out « "Parameters used in program:\n"
276         « "-------------------------------------------\n";
277 
278     for ( auto& itm: paramsUser ) {
279         out « indent « std::setw(max_len) « std::left « itm.paramName « " [";
280         switch (itm.paramType) {
281             case BoolType:
282                 out « std::boolalpha « *((bool *) (itm.paramPtr))
283                     « std::resetiosflags(out.flags());
284                 break;
285             case IntType:
286                 out « *((int*) itm.paramPtr);
287                 break;
288             case DoubleType:
289                 out « *((double*) itm.paramPtr);
290                 break;
291             case StringType:
292                 out « *((std::string*) itm.paramPtr);
293                 break;
294             case OutputType:
295                 out « *((Output *) (itm.paramPtr));
296                 break;
297         }
298         out « "]\n";
299     }
300     out « '\n';
301 }
```

9.11.2.8 PrintHelp()

```
void Parameters::PrintHelp (
            std::ostream & out = std::cout ) const
```

Print the help messages for Param.

Print out usage help for command line.

```
305 {
306     static const char *indent = "    ";
307     static const char *types[] = {"<bool>", "<int>", "<double>", "<string>", "<Output>"};
308 
309     out « "Usage: " « argv[0] « " [options] ...\n"
310         « "Options:\n" « indent « std::setw(21) « std::left
311         « "-h, --help" « "              : print help information and exit\n";
312 
313     for ( const auto& prm: paramsUser ) {
314         ParamType type = prm.paramType;
315         out « indent « std::setw(12) « std::left « prm.paramName
316             « " " « std::setw(8) « types[type];
317         if ( prm.paramMarker == 0)      out « ", optional   ";
318         else if ( prm.paramMarker == 1) out « ", required   ";
319         else                            out « ", params file";
320         if ( !prm.paramHelp.empty() ) out « " : " « prm.paramHelp;
321 
322         out « ", default = [";
323         switch ( type ) {
324             case BoolType:
325                 out « std::boolalpha « *(bool *) (prm.paramPtr)
326                     « std::setiosflags(out.flags());
```

```
327                 break;
328             case IntType:
329                 out « *(int *) (prm.paramPtr);
330                 break;
331             case DoubleType:
332                 out « *(double *) (prm.paramPtr);
333                 break;
334             case StringType:
335                 out « *(char **) (prm.paramPtr);
336                 break;
337             case OutputType:
338                 out « *(Output *) (prm.paramPtr);
339                 break;
340         }
341         out « "]\n";
342     }
343 }
```

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

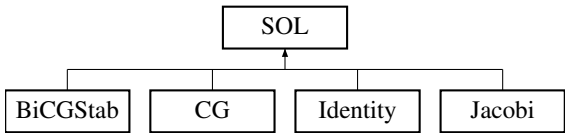
- Param.hxx
- Param.cxx

9.12 SOL Class Reference

Base class for iterative solvers.

`#include <SOL.hxx>`

Inheritance diagram for SOL:



Public Member Functions

- SOL ()
  Default constructor.
- ~SOL ()
  Default destructor.
- void SetOutput (Output verbose)
  Set output level.
- void SetMaxIter (int maxIter)
  Set max number of iterations.
- void SetMinIter (int minIter)
  Set min number of iterations.
- void SetSafeIter (int safeIter)
  Set number of safe-guard iterations.
- void SetRestart (int restart)
  Set restart number for Krylov methods.
- void SetRelTol (double relTol)
  Set tolerance for relative residual.
- void SetAbsTol (double absTol)

Set tolerance for absolute residual.

- void SetWeight (double alpha)

  Set weight for correction schemes.

- void SetSolType (SOLType solver)

  Set solver type.

- const char * GetSolType (SOLType type) const

  Get solver type.

- double GetNorm2 () const

  Get Euclidean norm of residual.

- double GetInfNorm () const

  Get infinity norm of residual.

- int GetIterations () const

  Get number of iterations.

- void PrintParam (std::ostream &out=std::cout) const

  Print parameters.

- void PrintHead (std::ostream &out=std::cout) const

  Print out iteration information table header.

- void PrintInfo (const int &iter, const double &resRel, const double &resAbs, const double &factor, std::ostream &out=std::cout) const

  Print out iteration information for iterative solvers.

- void PrintFinal (const int &iter, const double &resRel, const double &resAbs, const double &ratio, std::ostream &out=std::cout) const

  Print out final status of an iterative method.

- virtual void SetPC (SOL &pc)

  Setup preconditioner operator.

- virtual FaspRetCode Setup (const LOP &A)

  Setup the iterative method.

- virtual void Clean ()

  Release temporary memory and clean up.

- virtual FaspRetCode Solve (const VEC &b, VEC &x)

  Solve Ax=b using the iterative method.

Static Public Member Functions

- static void SetSolTypeFromName (SOLParams &params)

  Set solver type from its name.

Protected Member Functions

- void WarnRealRes (double relres) const

  Warning for actual relative residual.

- void WarnCompRes (double relres) const

  Warning for computed relative residual.

- void WarnDiffRes (double reldiff, double relres) const

  Output relative difference and residual.

Protected Attributes

- const LOP * A
  Coefficient matrix in Ax=b.
- SOL * pc
  Preconditioner for this solver.
- double norm2
  Euclidean norm.
- double normInf
  Infinity norm.
- int numIter
  Number of iterations when exit.
- SOLParams params
  solver parameters

9.12.1 Detailed Description

Base class for iterative solvers.

9.12.2 Member Function Documentation

9.12.2.1 GetInfNorm()

```
double SOL::GetInfNorm ( ) const
```

Get infinity norm of residual.

Get Inf norm of the residual vector.

```
207 {
208     return this->normInf;
209 }
```

References normInf.

9.12.2.2 GetIterations()

```
int SOL::GetIterations ( ) const
```

Get number of iterations.

Get the value of numIter.

```
213 {
214     return this->numIter;
215 }
```

References numIter.

9.12.2.3 GetNorm2()

```
double SOL::GetNorm2 ( ) const
```

Get Euclidean norm of residual.

Get L2 norm of the residual vector.

```
201 {
202     return this->norm2;
203 }
```

References norm2.

9.12.2.4 PrintHead()

```
void SOL::PrintHead (
            std::ostream & out = std::cout ) const
```

Print out iteration information table header.

Print out iteration information table head.

```
37 {
38     if ( params.verbose >= PRINT_MIN && params.minIter < params.maxIter ) {
39         out « "-------------------------------------------\n";
40         out « " It Num | ||r||/||b|| |    ||r||    |  Ratio \n";
41         out « "-------------------------------------------\n";
42     }
43 }
```

References params, and SOLParams::verbose.

9.12.2.5 SetAbsTol()

```
void SOL::SetAbsTol (
            double absTol )
```

Set tolerance for absolute residual.

Set value for absTol.

```
145 {
146     params.absTol = absTol;
147 }
```

References SOLParams::absTol, and params.

9.12.2.6 SetMaxIter()

```
void SOL::SetMaxIter (
            int maxIter )
```

Set max number of iterations.

Set value for maxIter.

```
115 {
116     params.maxIter = maxIter;
117 }
```

References SOLParams::maxIter, and params.

9.12.2.7 SetMinIter()

```
void SOL::SetMinIter (
            int minIter )
```

Set min number of iterations.

Set value for minIter.

```
121 {
122     params.minIter = minIter;
123 }
```

References SOLParams::minIter, and params.

9.12.2.8 SetOutput()

```
void SOL::SetOutput (
            Output verbose )
```

Set output level.

Set output level verbose.

```
109 {
110     params.verbose = verbose;
111 }
```

References params, and SOLParams::verbose.

9.12.2.9 SetPC()

```
void SOL::SetPC (
            SOL & pc )  [virtual]
```

Setup preconditioner operator.

Build preconditioner operator.

```
234 {
235     this->pc = &precond;
236 }
```

References pc.

9.12.2.10 SetRelTol()

```
void SOL::SetRelTol (
            double relTol )
```

Set tolerance for relative residual.

Set value for relTol.

```
139 {
140     params.relTol = relTol;
141 }
```

References params, and SOLParams::relTol.

9.12.2.11 SetRestart()

```
void SOL::SetRestart (
            int restart )
```

Set restart number for Krylov methods.

Set value for restart.

```
133 {
134     params.restart = restart;
135 }
```

References params, and SOLParams::restart.

9.12.2.12 SetSafeIter()

```
void SOL::SetSafeIter (
            int safeIter )
```

Set number of safe-guard iterations.

Set value for safeIter.

```
127 {
128     params.safeIter = safeIter;
129 }
```

References params, and SOLParams::safeIter.

9.12.2.13 SetSolType()

```
void SOL::SetSolType (
            SOLType solver )
```

Set solver type.

Set SOLType.

```
157 {
158     params.type = type;
159 }
```

References params, and SOLParams::type.

9.12.2.14 SetSolTypeFromName()

```
void SOL::SetSolTypeFromName (
            SOLParams & params )  [static]
```

Set solver type from its name.

Set value for SOLType using algName.

```
163 {
164     for ( char & c : params.algName ) c = std::tolower(c); // Change to lowercase
165     if ( params.algName == "cg" )
166         params.type = SOLType::CG;
167     else if ( params.algName == "bicgstab" )
168         params.type = SOLType::BICGSTAB;
169     else { // default solver type
170         params.type = SOLType::CG;
171         if ( params.verbose > PRINT_NONE )
172             FASPXX_WARNING("Unknown solver type. Using CG instead!")
173     }
174 }
```

References SOLParams::algName, BICGSTAB, CG, params, SOLParams::type, and SOLParams::verbose.

9.12.2.15 SetWeight()

```
void SOL::SetWeight (
            double alpha )
```

Set weight for correction schemes.

Set value for weight.

```
151 {
152     params.weight = alpha;
153 }
```

References params, and SOLParams::weight.

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

- SOL.hxx
- SOL.cxx

9.13 SOLParams Struct Reference

Iterative solver parameters.

```
#include <SOL.hxx>
```

Public Attributes

- SOLType type

  Algorithm type.
- string algName

  Algorithm name.
- int maxIter

  Maximal number of iterations.
- int minIter

  Minimal number of iterations.
- int safeIter

  Minimal number of iterations before safe-guard.
- int restart

  Restart number.
- double relTol

  Tolerance for relative residual.
- double absTol

  Tolerance for absolute residual.
- double weight

  Weight for correction schemes.
- Output verbose

  Output verbosity level.

9.13.1 Detailed Description

Iterative solver parameters.

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

- SOL.hxx

9.14 VEC Class Reference

General vector class.

`#include <VEC.hxx>`

Public Member Functions

- VEC ()

  Default constructor.
- VEC (const INT &size, const DBL &value=0.0)

  Construct a new VEC with the given size and a constant value.
- VEC (const std::vector< DBL > &src)

  Construct a new VEC by copying values from a vector.
- VEC (const VEC &src)

  Clone from another VEC.
- VEC (const INT &size, const DBL *src)

  Construct a new VEC by copying values from a pointer.
- ~VEC ()=default

  Default destructor.
- VEC & operator= (const VEC &v)

  Overload the = operator.
- DBL & operator[ ] (const INT &position)

  Overload the [] operator.
- const DBL & operator[ ] (const INT &position) const

  Overload the [] operator, entries cannot be modified.
- VEC & operator+= (const VEC &v)

  Overload += operator.
- VEC & operator-= (const VEC &v)

  Overload -= operator.
- void Reserve (const INT &size)

  Set the size of VEC object and reserve memory.
- void SetValues (const INT &size, const DBL &value=0.0)

  Assign the size and the same value to a VEC object.
- void SetValues (const std::vector< DBL > &src)

  Assign a vector object to a VEC object.
- void SetValues (const INT &size, const DBL *array)

  Assign values of a DBL array to a VEC object.
- DBL GetValue (const INT &position) const

  *Get the value of (*this)[position].*
- void GetValues (const INT &size, const INT *index, DBL *array) const

  Get multiple values and save them in an array.
- void GetArray (DBL **array)

  Get pointer to this->values.
- void GetArray (const DBL **array) const

  Get pointer to this->values, entries cannot be modified.
- INT GetSize () const

  *Get the size of *this.*
- void Scale (const DBL &a)

  Scale by a scalar.
- void Reciprocal ()

  Compute reciprocal pointwise.
- void PointwiseMult (const VEC &v)

  Scale by a vector pointwise.
- void PointwiseDivide (const VEC &v)

  Divide pointwise by a nonzero vector.
- void CopyTo (VEC &dst) const

Copy ∗this to another VEC.

- void Shift (const DBL &a)

  Shift by a scalar pointwise.
- void Abs ()

  Compute absolute values pointwise.
- void AXPY (const DBL &a, const VEC &x)

  y = a ∗ x + y.
- void XPAY (const DBL &a, const VEC &x)

  y = x + a ∗ y.
- void AXPBY (const DBL &a, const DBL &b, const VEC &y)

  x = a ∗ x + b ∗ y.
- void WAXPBY (const DBL &a, const VEC &x, const DBL &b, const VEC &y)

  ∗this = a ∗ v1 + b ∗ v2.
- DBL Max () const

  Find maximal value.
- DBL Min () const

  Find minimal value.
- DBL Norm2 () const

  Compute Euclidean norm.
- DBL NormInf () const

  Compute infinity norm.
- DBL Dot (const VEC &v) const

  Dot product of with v.

Friends

- class **MAT**

9.14.1 Detailed Description

General vector class.

9.14.2 Constructor & Destructor Documentation

9.14.2.1 VEC() [1/4]

```
VEC::VEC (
            const INT & size,
            const DBL & value = 0.0 )  [explicit]
```

Construct a new VEC with the given size and a constant value.

Assign the size and the same value to a VEC object.

```
17 {
18     this->values.assign(size, value);
19     this->size = size;
20 }
```

9.14.2.2 VEC() [2/4]

```
VEC::VEC (
            const std::vector< DBL > & src )  [explicit]
```

Construct a new VEC by copying values from a vector.

Assign a vector object to a VEC object.

```
24 {
25     this->values = src;
26     this->size = src.size();
27 }
```

9.14.2.3 VEC() [3/4]

```
VEC::VEC (
            const VEC & src )
```

Clone from another VEC.

Assign a const VEC object to a VEC object.

```
31 {
32     this->values = src.values;
33     this->size = src.size;
34 }
```

9.14.2.4 VEC() [4/4]

```
VEC::VEC (
            const INT & size,
            const DBL * src )  [explicit]
```

Construct a new VEC by copying values from a pointer.

Assign a DBL array to a VEC object. If source is nullptr, return an empty VEC.

```
38 {
39     if ( src == nullptr || size == 0 ) {
40         this->size = 0;
41         return;
42     }
43     this->values.assign(src, src + size);
44     this->size = size;
45 }
```

9.14.3 Member Function Documentation

9.14.3.1 Abs()

```
void VEC::Abs ( )
```

Compute absolute values pointwise.

(∗this)[i] = abs((∗this)[i]), unroll long for loops.

```
243 {
244     INT i;
245     const INT len = this->size - this->size % 4;
246     for ( i = 0; i < len; i += 4 ) {
247         this->values[i]     = fabs(this->values[i]);
248         this->values[i + 1] = fabs(this->values[i + 1]);
249         this->values[i + 2] = fabs(this->values[i + 2]);
250         this->values[i + 3] = fabs(this->values[i + 3]);
251     }
252     for ( i = len; i < this->size; ++i ) this->values[i] = fabs(this->values[i]);
253 }
```

9.14.3.2 AXPBY()

```
void VEC::AXPBY (
              const DBL & a,
              const DBL & b,
              const VEC & y )
```

x = a ∗ x + b ∗ y.

x = a ∗ x + b ∗ y, unroll long for loops.

```
286 {
287     INT i;
288     const INT len = this->size - this->size % 4;
289     switch ( (a == 1.0) + 2 * (b == 1.0) ) {
290         case 0:
291             for ( i = 0; i < len; i += 4 ) {
292                 this->values[i]     = a * this->values[i]     + b * y.values[i];
293                 this->values[i + 1] = a * this->values[i + 1] + b * y.values[i + 1];
294                 this->values[i + 2] = a * this->values[i + 2] + b * y.values[i + 2];
295                 this->values[i + 3] = a * this->values[i + 3] + b * y.values[i + 3];
296             }
297             for ( i = len; i < this->size; ++i )
298                 this->values[i] = a * this->values[i] + b * y.values[i];
299             break;
300 
301         case 1:
302             for ( i = 0; i < len; i += 4 ) {
303                 this->values[i]     += b * y.values[i];
304                 this->values[i + 1] += b * y.values[i + 1];
305                 this->values[i + 2] += b * y.values[i + 2];
306                 this->values[i + 3] += b * y.values[i + 3];
307             }
308             for ( i = len; i < this->size; ++i ) this->values[i] += b * y.values[i];
309             break;
310 
311         case 2:
312             for ( i = 0; i < len; i += 4 ) {
313                 this->values[i]     = a * this->values[i]     + y.values[i];
314                 this->values[i + 1] = a * this->values[i + 1] + y.values[i + 1];
315                 this->values[i + 2] = a * this->values[i + 2] + y.values[i + 2];
316                 this->values[i + 3] = a * this->values[i + 3] + y.values[i + 3];
317             }
318             for ( i = len; i < this->size; ++i )
319                 this->values[i] = a * this->values[i] + y.values[i];
320             break;
321 
322         case 3:
323             for ( i = 0; i < len; i += 4 ) {
324                 this->values[i]     += y.values[i];
325                 this->values[i + 1] += y.values[i + 1];
326                 this->values[i + 2] += y.values[i + 2];
327                 this->values[i + 3] += y.values[i + 3];
328             }
329             for ( i = len; i < this->size; ++i ) this->values[i] += y.values[i];
330     }
331 }
```

9.14.3.3 AXPY()

```
void VEC::AXPY (
              const DBL & a,
              const VEC & x )
```

y = a ∗ x + y.

y = a ∗ x + y, unroll long for loops.

```
257 {
258     INT i;
259     const INT len = this->size - this->size % 4;
260     for ( i = 0; i < len; i += 4 ) {
261         this->values[i]     += a * x.values[i];
262         this->values[i + 1] += a * x.values[i + 1];
263         this->values[i + 2] += a * x.values[i + 2];
264         this->values[i + 3] += a * x.values[i + 3];
265     }
266     for ( i = len; i < this->size; ++i ) this->values[i] += a * x.values[i];
267 }
```

9.14.3.4 CopyTo()

```
void VEC::CopyTo (
              VEC & dst ) const
```

Copy ∗this to another VEC.

dst = ∗this.

```
222 {
223     dst.values = this->values;
224     dst.size = this->size;
225 }
```

9.14.3.5 Dot()

```
DBL VEC::Dot (
              const VEC & v ) const
```

Dot product of with v.

Dot product of with v, unroll long for loops.

```
474 {
475     INT i;
476     DBL dot1 = 0.0, dot2 = 0.0, dot3 = 0.0, dot4 = 0.0;
477     const INT len = this->size - this->size % 4;
478     for ( i = 0; i < len; i += 4 ) {
479         dot1 += this->values[i]     * v.values[i];
480         dot2 += this->values[i + 1] * v.values[i + 1];
481         dot3 += this->values[i + 2] * v.values[i + 2];
482         dot4 += this->values[i + 3] * v.values[i + 3];
483     }
484     for ( i = len; i < this->size; ++i ) dot1 += this->values[i] * v.values[i];
485     return (dot1 + dot2 + dot3 + dot4);
486 }
```

9.14.3.6 GetArray() [1/2]

```
void VEC::GetArray (
            const DBL ** array ) const
```

Get pointer to this->values, entries cannot be modified.

The pointer array points this->values and it can be used to access data of VEC. The values cannot be modified.

```
154 {
155     *array = this->values.data();
156 }
```

9.14.3.7 GetArray() [2/2]

```
void VEC::GetArray (
            DBL ** array )
```

Get pointer to this->values.

The pointer array points this->values and it can be used to access data of VEC.

```
147 {
148     *array = this->values.data();
149 }
```

9.14.3.8 GetSize()

```
INT VEC::GetSize ( ) const
```

Get the size of *this.

Return the size of VEC.

```
160 {
161     return this->size;
162 }
```

9.14.3.9 GetValue()

```
DBL VEC::GetValue (
            const INT & position ) const
```

Get the value of (*this)[position].

Return the value of (*this)[position].

```
131 {
132     return this->values.at(position);
133 }
```

9.14.3.10 GetValues()

```
void VEC::GetValues (
             const INT & size,
             const INT * index,
             DBL * array ) const
```

Get multiple values and save them in an array.

Get value of this->values[index[j] and save it in array[j].

Note

Users should allocate memory for array before calling this function!

```
137 {
138     if ( size == 0 || this->size == 0 ) {
139         array = nullptr;
140         return;
141     }
142     for ( INT j = 0; j < size; ++j ) array[j] = this->values[index[j] % this->size];
143 }
```

9.14.3.11 Max()

```
DBL VEC::Max ( ) const
```

Find maximal value.

Find maximal value, unroll long for loops.

```
388 {
389     DBL max1 = SMALL, max2 = SMALL, max3 = SMALL, max4 = SMALL;
390
391     INT i;
392     const INT len = this->size - this->size % 4;
393     for ( i = 0; i < len; i += 4 ) {
394         if ( max1 < this->values[i] )     max1 = this->values[i];
395         if ( max2 < this->values[i + 1] ) max2 = this->values[i + 1];
396         if ( max3 < this->values[i + 2] ) max3 = this->values[i + 2];
397         if ( max4 < this->values[i + 3] ) max4 = this->values[i + 3];
398     }
399     for ( i = len; i < this->size; ++i )
400         if ( max1 < this->values[i] ) max1 = this->values[i];
401
402     max1 = max1 >= max2 ? max1 : max2;
403     max3 = max3 >= max4 ? max3 : max4;
404     return max1 >= max3 ? max1 : max3;
405 }
```

References SMALL.

9.14.3.12 Min()

```
DBL VEC::Min ( ) const
```

Find minimal value.

Find min(∗this), unroll long for loops.

```
409 {
410     DBL min1 = LARGE, min2 = LARGE, min3 = LARGE, min4 = LARGE;
411
412     INT i;
413     const INT len = this->size - this->size % 4;
414     for ( i = 0; i < len; i += 4 ) {
415         if (min1 > this->values[i])     min1 = this->values[i];
416         if (min2 > this->values[i + 1]) min2 = this->values[i + 1];
417         if (min3 > this->values[i + 2]) min3 = this->values[i + 2];
418         if (min4 > this->values[i + 3]) min4 = this->values[i + 3];
419     }
420     for ( i = len; i < this->size; ++i )
421         if (min1 > this->values[i]) min1 = this->values[i];
422
423     min1 = min1 <= min2 ? min1 : min2;
424     min3 = min3 <= min4 ? min3 : min4;
425     return min1 <= min3 ? min1 : min3;
426 }
```

References LARGE.

9.14.3.13 Norm2()

```
DBL VEC::Norm2 ( ) const
```

Compute Euclidean norm.

Compute Euclidean norm of ∗this, unroll long for loops.

```
430 {
431     INT i;
432     DBL tmp1 = 0.0, tmp2 = 0.0, tmp3 = 0.0, tmp4 = 0.0;
433     const INT len = this->size - this->size % 4;
434     for ( i = 0; i < len; i += 4 ) {
435         tmp1 += std::pow(this->values[i], 2);
436         tmp2 += std::pow(this->values[i + 1], 2);
437         tmp3 += std::pow(this->values[i + 2], 2);
438         tmp4 += std::pow(this->values[i + 3], 2);
439     }
440     for ( i = len; i < this->size; ++i ) tmp1 += std::pow(this->values[i], 2);
441
442     return sqrt(tmp1 + tmp2 + tmp3 + tmp4);
443 }
```

9.14.3.14 NormInf()

```
DBL VEC::NormInf ( ) const
```

Compute infinity norm.

Compute infinity norm of ∗this, unroll long for loops.

```
447 {
448     INT i;
449     DBL tmpNorm1 = 0.0, tmpNorm2 = 0.0, tmpNorm3 = 0.0, tmpNorm4 = 0.0;
450     DBL tmp1, tmp2, tmp3, tmp4;
451     const INT len = this->size - this->size % 4;
452     for ( i = 0; i < len; i += 4 ) {
453         tmp1 = fabs(this->values[i]);
```

```
454         if ( tmp1 > tmpNorm1 ) tmpNorm1 = tmp1;
455         tmp2 = fabs(this->values[i + 1]);
456         if ( tmp2 > tmpNorm2 ) tmpNorm2 = tmp2;
457         tmp3 = fabs(this->values[i + 2]);
458         if ( tmp3 > tmpNorm3 ) tmpNorm3 = tmp3;
459         tmp4 = fabs(this->values[i + 3]);
460         if ( tmp4 > tmpNorm4 ) tmpNorm4 = tmp4;
461     }
462     for ( i = len; i < this->size; ++i ) {
463         tmp1 = fabs(this->values[i]);
464         if ( tmp1 > tmpNorm1 ) tmpNorm1 = tmp1;
465     }
466
467     tmpNorm1 = tmpNorm1 >= tmpNorm2 ? tmpNorm1 : tmpNorm2;
468     tmpNorm3 = tmpNorm3 >= tmpNorm4 ? tmpNorm3 : tmpNorm4;
469     return (tmpNorm1 > tmpNorm3 ? tmpNorm1 : tmpNorm3);
470 }
```

9.14.3.15 operator+=()

```
VEC & VEC::operator+= (
            const VEC & v )
```

Overload += operator.

Unroll for loops to speed up calculation.

```
69 {
70     INT i;
71     const INT len = this->size - this->size % 4;
72     for ( i = 0; i < len; i += 4 ) {
73         this->values[i]     += v.values[i];
74         this->values[i + 1] += v.values[i + 1];
75         this->values[i + 2] += v.values[i + 2];
76         this->values[i + 3] += v.values[i + 3];
77     }
78     for ( i = len; i < this->size; ++i ) this->values[i] += v.values[i];
79     return *this;
80 }
```

9.14.3.16 operator-=()

```
VEC & VEC::operator-= (
            const VEC & v )
```

Overload -= operator.

Unroll for loops to speed up calculation.

```
84 {
85     INT i;
86     const INT len = this->size - this->size % 4;
87     for ( i = 0; i < len; i += 4 ) {
88         this->values[i]     -= v.values[i];
89         this->values[i + 1] -= v.values[i + 1];
90         this->values[i + 2] -= v.values[i + 2];
91         this->values[i + 3] -= v.values[i + 3];
92     }
93     for ( i = len; i < this->size; ++i ) this->values[i] -= v.values[i];
94     return *this;
95 }
```

9.14.3.17 operator=()

```
VEC & VEC::operator= (
            const VEC & v )
```

Overload the = operator.

Assignment for the VEC object.

```
49 {
50     this->values = src.values;
51     this->size = src.size;
52     return *this;
53 }
```

9.14.3.18 operator[]() [1/2]

```
DBL & VEC::operator[] (
            const INT & position )
```

Overload the [] operator.

Regular [] operator, same behavior as array.

```
57 {
58     return this->values[position];
59 }
```

9.14.3.19 operator[]() [2/2]

```
const DBL & VEC::operator[] (
            const INT & position ) const
```

Overload the [] operator, entries cannot be modified.

Const [] operator, entries cannot be modified.

```
63 {
64     return this->values[position];
65 }
```

9.14.3.20 PointwiseDivide()

```
void VEC::PointwiseDivide (
            const VEC & v )
```

Divide pointwise by a nonzero vector.

Divide by a nonzero vector (*this)[i] = (*this)[i] / v[i], unroll long for loops.

```
208 {
209     INT i;
210     const INT len = this->size - this->size % 4;
211     for ( i = 0; i < len; i += 4 ) {
212         this->values[i]     /= v.values[i];
213         this->values[i + 1] /= v.values[i + 1];
214         this->values[i + 2] /= v.values[i + 2];
215         this->values[i + 3] /= v.values[i + 3];
216     }
217     for ( i = len; i < this->size; ++i ) this->values[i] /= v.values[i];
218 }
```

9.14.3.21 PointwiseMult()

```
void VEC::PointwiseMult (
            const VEC & v )
```

Scale by a vector pointwise.

(*this)[j] *= v[j], unroll long for loops.

```
180 {
181     INT i;
182     const INT len = this->size - this->size % 4;
183     for ( i = 0; i < len; i += 4 ) {
184         this->values[i]     *= v.values[i];
185         this->values[i + 1] *= v.values[i + 1];
186         this->values[i + 2] *= v.values[i + 2];
187         this->values[i + 3] *= v.values[i + 3];
188     }
189     for ( i = len; i < this->size; ++i ) this->values[i] *= v.values[i];
190 }
```

9.14.3.22 Reciprocal()

```
void VEC::Reciprocal ( )
```

Compute reciprocal pointwise.

(*this)[i] = 1 / (*this)[i], unroll long for loops.

```
194 {
195     INT i;
196     const INT len = this->size - this->size % 4;
197     for ( i = 0; i < len; i += 4 ) {
198         this->values[i]     = 1.0 / this->values[i];
199         this->values[i + 1] = 1.0 / this->values[i + 1];
200         this->values[i + 2] = 1.0 / this->values[i + 2];
201         this->values[i + 3] = 1.0 / this->values[i + 3];
202     }
203     for ( i = len; i < this->size; ++i ) this->values[i] = 1 / this->values[i];
204 }
```

9.14.3.23 Reserve()

```
void VEC::Reserve (
            const INT & size )
```

Set the size of VEC object and reserve memory.

Reserve memory for the vector values without changing the size.

```
99 {
100     this->values.reserve(size);
101 }
```

9.14.3.24 Scale()

```
void VEC::Scale (
            const DBL & a )
```

Scale by a scalar.

(∗this)[j] = a ∗ (∗this)[j], unroll long for loops.

```
166 {
167     INT i;
168     const INT len = this->size - this->size % 4;
169     for ( i = 0; i < len; i += 4 ) {
170         this->values[i]     *= a;
171         this->values[i + 1] *= a;
172         this->values[i + 2] *= a;
173         this->values[i + 3] *= a;
174     }
175     for ( i = len; i < this->size; ++i ) this->values[i] *= a;
176 }
```

9.14.3.25 SetValues() [1/3]

```
void VEC::SetValues (
            const INT & size,
            const DBL & value = 0.0 )
```

Assign the size and the same value to a VEC object.

Assign a single value to a VEC object.

```
105 {
106     this->size = size;
107     this->values.assign(size, value);
108 }
```

9.14.3.26 SetValues() [2/3]

```
void VEC::SetValues (
            const INT & size,
            const DBL * array )
```

Assign values of a DBL array to a VEC object.

Assign a DBL array to a VEC object. If source is nullptr, return an empty VEC.

```
119 {
120     if ( array == nullptr || size == 0 ) {
121         this->values.resize(0);
122         this->size = 0;
123         return;
124     }
125     this->values.assign(array, array + size);
126     this->size = size;
127 }
```

9.14.3.27 SetValues() [3/3]

```
void VEC::SetValues (
            const std::vector< DBL > & src )
```

Assign a vector object to a VEC object.

Assign vector values to a VEC object.

```
112 {
113     this->values = src;
114     this->size = src.size();
115 }
```

9.14.3.28 Shift()

```
void VEC::Shift (
            const DBL & a )
```

Shift by a scalar pointwise.

(∗this)[i] += a, unroll long for loops.

```
229 {
230     INT i;
231     const INT len = this->size - this->size % 4;
232     for ( i = 0; i < len; i += 4 ) {
233         this->values[i]     += a;
234         this->values[i + 1] += a;
235         this->values[i + 2] += a;
236         this->values[i + 3] += a;
237     }
238     for ( i = len; i < this->size; ++i ) this->values[i] += a;
239 }
```

9.14.3.29 WAXPBY()

```
void VEC::WAXPBY (
            const DBL & a,
            const VEC & x,
            const DBL & b,
            const VEC & y )
```

∗this = a ∗ v1 + b ∗ v2.

∗this = a ∗ v1 + b ∗ v2, unroll long for loops.

```
335 {
336     INT i;
337     this->size = v1.size;
338     const INT len = this->size - this->size % 4;
339     switch ( (a == 1) + 2 * (b == 1) ) {
340         case 0:
341             for ( i = 0; i < len; i += 4 ) {
342                 this->values[i]     = a * v1.values[i]     + b * v2.values[i];
343                 this->values[i + 1] = a * v1.values[i + 1] + b * v2.values[i + 1];
344                 this->values[i + 2] = a * v1.values[i + 2] + b * v2.values[i + 2];
345                 this->values[i + 3] = a * v1.values[i + 3] + b * v2.values[i + 3];
346             }
347             for ( i = len; i < this->size; ++i )
348                 this->values[i] = a * v1.values[i] + b * v2.values[i];
349             break;
350
351         case 1:
352             for ( i = 0; i < len; i += 4 ) {
```

```
353                 this->values[i]     = v1.values[i]     + b * v2.values[i];
354                 this->values[i + 1] = v1.values[i + 1] + b * v2.values[i + 1];
355                 this->values[i + 2] = v1.values[i + 2] + b * v2.values[i + 2];
356                 this->values[i + 3] = v1.values[i + 3] + b * v2.values[i + 3];
357             }
358             for ( i = len; i < this->size; ++i )
359                 this->values[i] = v1.values[i] + b * v2.values[i];
360             break;
361 
362         case 2:
363             for ( i = 0; i < len; i += 4 ) {
364                 this->values[i]     = a * v1.values[i]     + v2.values[i];
365                 this->values[i + 1] = a * v1.values[i + 1] + v2.values[i + 1];
366                 this->values[i + 2] = a * v1.values[i + 2] + v2.values[i + 2];
367                 this->values[i + 3] = a * v1.values[i + 3] + v2.values[i + 3];
368             }
369             for ( i = len; i < this->size; ++i )
370                 this->values[i] = a * v1.values[i] + v2.values[i];
371             break;
372 
373         case 3:
374             for ( i = 0; i < len; i += 4 ) {
375                 this->values[i]     = v1.values[i]     + v2.values[i];
376                 this->values[i + 1] = v1.values[i + 1] + v2.values[i + 1];
377                 this->values[i + 2] = v1.values[i + 2] + v2.values[i + 2];
378                 this->values[i + 3] = v1.values[i + 3] + v2.values[i + 3];
379             }
380             for ( i = len; i < this->size; ++i )
381                 this->values[i] = v1.values[i] + v2.values[i];
382             break;
383     }
384 }
```

9.14.3.30 XPAY()

```
void VEC::XPAY (
              const DBL & a,
              const VEC & x )
```

y = x + a ∗ y.

y = x + a ∗ y, unroll long for loops.

```
271 {
272     INT i;
273     const INT len = this->size - this->size % 4;
274     for ( i = 0; i < len; i += 4 ) {
275         this->values[i]     = a * this->values[i]     + x.values[i];
276         this->values[i + 1] = a * this->values[i + 1] + x.values[i + 1];
277         this->values[i + 2] = a * this->values[i + 2] + x.values[i + 2];
278         this->values[i + 3] = a * this->values[i + 3] + x.values[i + 3];
279     }
280     for ( i = len; i < this->size; ++i )
281         this->values[i] = x.values[i] + a * this->values[i];
282 }
```

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

- VEC.hxx
- VEC.cxx

Chapter 10

File Documentation

10.1 BiCGStab.hxx File Reference

Preconditioned BiCGStab class declaration.

```
#include <cmath>
#include <cfloat>
#include "ErrorLog.hxx"
#include "LOP.hxx"
#include "MAT.hxx"
#include "SOL.hxx"
```

Classes

- class BiCGStab

  Preconditioned bi-conjugate gradient stabilized method.

Macros

- #define __BICGSTAB_HEADER__

10.1.1 Detailed Description

Preconditioned BiCGStab class declaration.

Author

Kailei Zhang, Chensong Zhang

Date

Nov/25/2019

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

Author

Kailei Zhang, Chensong Zhang

Date

Nov/25/2019

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

10.1.2 Macro Definition Documentation

10.1.2.1 __BICGSTAB_HEADER__

```
#define __BICGSTAB_HEADER__
```

indicate BiCGStab.hxx has been included before

10.2 CG.cxx File Reference

Preconditioned CG class definition.

```
#include "Iter.hxx"
#include "CG.hxx"
```

10.2.1 Detailed Description

Preconditioned CG class definition.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/13/2019

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

10.3 CG.hxx File Reference

Preconditioned CG class declaration.

```
#include <cmath>
#include "ErrorLog.hxx"
#include "LOP.hxx"
#include "MAT.hxx"
#include "SOL.hxx"
```

Classes

- class CG

  Preconditioned conjugate gradient method.

Macros

- #define __CG_HEADER__

10.3.1 Detailed Description

Preconditioned CG class declaration.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/11/2019

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

10.3.2 Macro Definition Documentation

10.3.2.1 __CG_HEADER__

```
#define __CG_HEADER__
```

indicate CG.hxx has been included before

10.4 doxygen.hxx File Reference

Main page for Doxygen documentation.

Macros

- #define __DOXYGEN_HXX__

10.4.1 Detailed Description

Main page for Doxygen documentation.

Author

Chensong Zhang

Date

Sep/29/2019

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

10.4.2 Macro Definition Documentation

10.4.2.1 __DOXYGEN_HXX__

```
#define __DOXYGEN_HXX__
```

indicate doxygen.hxx has been included before

10.5 ErrorLog.hxx File Reference

Logging error and warning messages.

```
#include <sstream>
#include <iomanip>
#include <iostream>
```

Macros

- #define __ERRORLOG_HXX__
- #define _FASPXX_LOCATION_

  Print out location at (file, line) and function name.
- #define _FASPXX_MASSAGE_(msg)

  Log error messages.
- #define FASPXX_WARNING(msg)

  Log warning messages.
- #define FASPXX_ABORT(msg)

  Abort if critical error happens.
- #define FASPXX_ASSERT(cond, msg)

  Check condition and log user messages.

10.5.1 Detailed Description

Logging error and warning messages.

Author

Ronghong Fan

Date

Nov/01/2019

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

10.5.2 Macro Definition Documentation

10.5.2.1 __ERRORLOG_HXX__

```
#define __ERRORLOG_HXX__
```

indicate ErrorLog.hxx has been included before

10.5.2.2 _FASPXX_LOCATION_

```
#define _FASPXX_LOCATION_
```

Value:

```
    "\n    --> function: " « __PRETTY_FUNCTION__ «      \
    "\n    --> file:     " « __FILE__ « ':' « __LINE__
```

Print out location at (file, line) and function name.

10.5.2.3 _FASPXX_MASSAGE_

```
#define _FASPXX_MASSAGE_(
              msg )
```

Value:

```
    {                                                \
        std::ostringstream info;                     \
        info « std::setprecision(16);               \
        info « msg « _FASPXX_LOCATION_ « '\n';  \
        std::cout « info.str().c_str();              \
    }
```

Log error messages.

10.5.2.4 FASPXX_ABORT

```
#define FASPXX_ABORT(
              msg )
```

Value:

```
    {                                                \
        _FASPXX_MASSAGE_("### ABORT: " « msg);     \
        std::abort();                                \
    }
```

Abort if critical error happens.

10.5.2.5 FASPXX_ASSERT

```
#define FASPXX_ASSERT(
              cond,
              msg )
```

Value:

```
   if (!(cond)) {                                                          \
      _FASPXX_MASSAGE_("### ASSERT: " « msg « " (" « #cond « ")"); \
   }
```

Check condition and log user messages.

10.5.2.6 FASPXX_WARNING

```
#define FASPXX_WARNING(
              msg )
```

Value:

```
    {                                                \
        _FASPXX_MASSAGE_("### WARNING: " « (msg));\
    }
```

Log warning messages.

10.6 faspxx.hxx File Reference

Main FASP++ header file.

Macros

- #define __FASPXX_HEADER__

Typedefs

- typedef unsigned int INT

  Index type: Must be non-negative!

- typedef double DBL

  Double precision numbers.

Variables

- const DBL SMALL_TOL = 1e-14

  Small positive real for tolerance.
- const DBL LARGE = 1e+60

  Largest double number.
- const DBL SMALL = -1e+60

  Smallest double number.
- const DBL CLOSE_ZERO = 1e-20

  Tolerance for closeness to zero.
- const DBL KSM_CHK_RATIO = 0.95

  Check ratio for Krylov space methods.
- const int MAX_STAG_NUM = 20

  Maximal number of stagnation checks.
- const int PRT_STEP_NUM = 20

  Print iteration info every N steps.

10.6.1 Detailed Description

Main FASP++ header file.

Author

Kailei Zhang

Date

Sep/01/2019

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

10.6.2 Macro Definition Documentation

10.6.2.1 __FASPXX_HEADER__

```
#define __FASPXX_HEADER__
```

indicate faspxx.hxx has been included before

10.7 Iter.cxx File Reference

Simple iterative methods definition.

```
#include "Iter.hxx"
```

10.7.1 Detailed Description

Simple iterative methods definition.

Author

Chensong Zhang, Kailei Zhang

Date

Dec/02/2019

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

10.8 Iter.hxx File Reference

Simple iterative methods declaration.

```
#include <cmath>
#include "faspxx.hxx"
#include "SOL.hxx"
#include "MAT.hxx"
```

Classes

- class Identity

  Identity operator.
- class Jacobi

  Jacobi iterator.

Macros

- #define __ITER_HEADER__

10.8.1 Detailed Description

Simple iterative methods declaration.

Author

Chensong Zhang, Kailei Zhang

Date

Dec/02/2019

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

10.8.2 Macro Definition Documentation

10.8.2.1 __ITER_HEADER__

```
#define __ITER_HEADER__
```

indicate Iter.hxx has been included before

10.9 Krylov.cxx File Reference

General interface for Krylov subspace methods.

```
#include "SOL.hxx"
#include "Krylov.hxx"
```

Functions

- FaspRetCode Krylov (LOP &A, VEC &b, VEC &x, SOL &pc, SOLParams &params)

  All supported Krylov methods can be accessed using this interface.

10.9.1 Detailed Description

General interface for Krylov subspace methods.

Author

Chensong Zhang, Kailei Zhang

Date

Dec/27/2019

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

10.9.2 Function Documentation

10.9.2.1 Krylov()

```
FaspRetCode Krylov (
             LOP & A,
             VEC & b,
             VEC & x,
             SOL & pc,
             SOLParams & params )
```

All supported Krylov methods can be accessed using this interface.
General interface to Krylov subspace methods.

```
17 {
18     FaspRetCode retCode = FaspRetCode::SUCCESS;
19
20     SOL solver;
21     solver.SetSolTypeFromName(params); // get solver type
22     auto sol = &solver;
23
24     switch (params.type) {
25         case SOLType::CG :
26             sol = new class CG();
27             sol->SetOutput(params.verbose);
28             sol->SetMaxIter(params.maxIter);
29             sol->SetMinIter(params.minIter);
30             sol->SetRestart(params.restart);
31             sol->SetRelTol(params.relTol);
32             sol->SetAbsTol(params.absTol);
33             sol->SetSafeIter(params.safeIter);
34             sol->Setup(A);
35             sol->SetPC(pc);
36             retCode = sol->Solve(b, x);
37             break;
38         case SOLType::BICGSTAB :
39             sol = new class BiCGStab();
40             sol->SetOutput(params.verbose);
41             sol->SetMaxIter(params.maxIter);
42             sol->SetMinIter(params.minIter);
43             sol->SetRestart(params.restart);
44             sol->SetRelTol(params.relTol);
45             sol->SetAbsTol(params.absTol);
46             sol->SetSafeIter(params.safeIter);
47             sol->Setup(A);
48             sol->SetPC(pc);
49             retCode = sol->Solve(b, x);
50             break;
51         default: // should never reach here!!!
52             if ( params.verbose > PRINT_NONE )
53                 FASPXX_WARNING("Unknown Krylov method type")
54             std::cout « sol->GetSolType(params.type) « "is not supported!\n";
55     }
56
57     return retCode;
58 }
```

References SOL::A, SOLParams::absTol, BICGSTAB, CG, SOLParams::maxIter, SOLParams::minIter, SOL::params, SOL::pc, SOLParams::relTol, SOLParams::restart, SOLParams::safeIter, SOL::SetOutput(), SOL::SetSolTypeFromName(), SUCCESS, SOLParams::type, and SOLParams::verbose.

10.10 Krylov.hxx File Reference

Declaration of interface to general Krylov subspace methods.

```
#include "RetCode.hxx"
#include "SOL.hxx"
#include "Iter.hxx"
#include "CG.hxx"
#include "BiCGStab.hxx"
```

Macros

- #define __KRYLOV_HEADER__

Functions

- FaspRetCode Krylov (LOP &A, VEC &b, VEC &x, SOL &pc, SOLParams &params)

  General interface to Krylov subspace methods.

10.10.1 Detailed Description

Declaration of interface to general Krylov subspace methods.

Author

Chensong Zhang, Kailei Zhang

Date

Dec/27/2019

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

10.10.2 Macro Definition Documentation

10.10.2.1 __KRYLOV_HEADER__

```
#define __KRYLOV_HEADER__
```

indicate Krylov.hxx has been included before

10.10.3 Function Documentation

10.10.3.1 Krylov()

```
FaspRetCode Krylov (
              LOP & A,
              VEC & b,
              VEC & x,
              SOL & pc,
              SOLParams & params )
```

General interface to Krylov subspace methods.

General interface to Krylov subspace methods.

```
17 {
18     FaspRetCode retCode = FaspRetCode::SUCCESS;
19
20     SOL solver;
```

```
21      solver.SetSolTypeFromName(params); // get solver type
22      auto sol = &solver;
23
24      switch (params.type) {
25          case SOLType::CG :
26              sol = new class CG();
27              sol->SetOutput(params.verbose);
28              sol->SetMaxIter(params.maxIter);
29              sol->SetMinIter(params.minIter);
30              sol->SetRestart(params.restart);
31              sol->SetRelTol(params.relTol);
32              sol->SetAbsTol(params.absTol);
33              sol->SetSafeIter(params.safeIter);
34              sol->Setup(A);
35              sol->SetPC(pc);
36              retCode = sol->Solve(b, x);
37              break;
38          case SOLType::BICGSTAB :
39              sol = new class BiCGStab();
40              sol->SetOutput(params.verbose);
41              sol->SetMaxIter(params.maxIter);
42              sol->SetMinIter(params.minIter);
43              sol->SetRestart(params.restart);
44              sol->SetRelTol(params.relTol);
45              sol->SetAbsTol(params.absTol);
46              sol->SetSafeIter(params.safeIter);
47              sol->Setup(A);
48              sol->SetPC(pc);
49              retCode = sol->Solve(b, x);
50              break;
51          default: // should never reach here!!!
52              if ( params.verbose > PRINT_NONE )
53                  FASPXX_WARNING("Unknown Krylov method type")
54              std::cout << sol->GetSolType(params.type) << "is not supported!\n";
55      }
56
57      return retCode;
58  }
```

References SOL::A, SOLParams::absTol, BICGSTAB, CG, SOLParams::maxIter, SOLParams::minIter, SOL::params, SOL::pc, SOLParams::relTol, SOLParams::restart, SOLParams::safeIter, SOL::SetOutput(), SOL::SetSolTypeFromName(), SUCCESS, SOLParams::type, and SOLParams::verbose.

10.11 LOP.cxx File Reference

Linear operator class definition.

```
#include "LOP.hxx"
```

10.11.1 Detailed Description

Linear operator class definition.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/27/2019

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

10.12 LOP.hxx File Reference

Linear operator class declaration.

```
#include <vector>
#include "faspxx.hxx"
#include "ErrorLog.hxx"
#include "VEC.hxx"
```

Classes

- class LOP

  Linear operator virtual class.

Macros

- #define __LOP_HEADER__

10.12.1 Detailed Description

Linear operator class declaration.

Author

Chensong Zhang

Date

Sep/27/2019

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

10.12.2 Macro Definition Documentation

10.12.2.1 __LOP_HEADER__

```
#define __LOP_HEADER__
```

indicate LOP.hxx has been included before

10.13 MAT.cxx File Reference

Definition of the default matrix class.

```
#include <fstream>
#include "MAT.hxx"
#include "MATUtil.hxx"
```

Functions

- void WriteCSR (char *filename, MAT mat)

  Write data to a disk file in CSR format.

- void WriteMTX (char *filename, MAT mat)

  Write data to a disk file in MTX format.

10.13.1 Detailed Description

Definition of the default matrix class.

Author

Kailei Zhang

Date

Sep/25/2019

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

10.13.2 Function Documentation

10.13.2.1 WriteCSR()

```
void WriteCSR (
            char * filename,
            MAT mat )
```

Write data to a disk file in CSR format.

Write an MAT matrix to a disk file in CSR format.

```
753                                         {
754     std::ofstream out;
755     out.open(filename);
756 
757     out « mat.nrow « " " « mat.mcol « " " « mat.nnz « "\n";
758     for (INT j = 0; j < mat.nrow + 1; ++j) out « mat.rowPtr[j] « "\n";
759     for (INT j = 0; j < mat.nnz; ++j) out « mat.colInd[j] « "\n";
760     for (INT j = 0; j < mat.nnz; ++j) out « mat.values[j] « "\n";
761 
762     out.close();
763 }
```

References LOP::mcol, and LOP::nrow.

10.13.2.2 WriteMTX()

```
void WriteMTX (
            char * filename,
            MAT mat )
```

Write data to a disk file in MTX format.

Write an MAT matrix to a disk file in MTX format.

```
766                                         {
767     INT begin, end, j, k;
768     std::ofstream out;
769     out.open(filename);
770     MAT tmp = mat;
771     tmp.Transpose();
772 
773     out « tmp.nrow « " " « tmp.mcol « " " « tmp.nnz « "\n";
774     for (j = 0; j < tmp.nrow; ++j) {
775         begin = tmp.rowPtr[j];
776         end = tmp.rowPtr[j + 1];
777         for (k = begin; k < end; ++k)
778             out « j « " " « tmp.colInd[j] « " " « tmp.values[j] « std::endl;
779     }
780 
781     out.close();
782 }
```

References LOP::mcol, LOP::nrow, and MAT::Transpose().

10.14 MAT.hxx File Reference

Matrix class declaration.

```
#include <vector>
#include "faspxx.hxx"
#include "VEC.hxx"
#include "LOP.hxx"
```

Classes

- class MAT

  Sparse matrix class.

Macros

- #define __MAT_HEADER__

10.14.1 Detailed Description

Matrix class declaration.

Author

Kailei Zhang, Chensong Zhang

Date

Sep/25/2019

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

10.14.2 Macro Definition Documentation

10.14.2.1 __MAT_HEADER__

```
#define __MAT_HEADER__
```

indicate MAT.hxx has been included before

10.15 MATUtil.cxx File Reference

Some auxiliary functions for MAT.

```
#include "MATUtil.hxx"
#include "RetCode.hxx"
```

Functions

- FaspRetCode CheckMATAddSize (const MAT &mat1, const MAT &mat2)

  Check whether two matrices have same sizes for addition.
- FaspRetCode CheckMATMultSize (const MAT &mat1, const MAT &mat2)

  Check MAT-MAT multiplication sizes.
- FaspRetCode CheckMATSize (const MAT &mat, const INT &row, const INT &col)

  Check whether (row,col) is out of bound.
- FaspRetCode CheckMATRowSize (const MAT &mat, const INT &row)

  Check whether (row,:) is out of bound.
- FaspRetCode CheckMATColSize (const MAT &mat, const INT &col)

  Check whether (:,col) is out of bound.
- FaspRetCode CheckMATVECSize (const MAT &mat, const VEC &vec)

  Check MAT-VEC multiplication sizes.
- FaspRetCode CheckCSR (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr)

  Check whether the data is good for CSR.
- FaspRetCode CheckCSRx (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, const std::vector< INT > &diagPtr)

  Check whether the data is good for CSRx.

- FaspRetCode CSRtoMAT (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, MAT &mat)

  Convert a CSR matrix to MAT (private)
- FaspRetCode MTXtoMAT (const INT &row, const INT &col, const INT &nnz, const std::vector< INT > &rowInd, const std::vector< INT > &colInd, const std::vector< DBL > &values, MAT &mat)

  Convert MTX data to MAT.
- FaspRetCode SortCSRRow (const INT &row, const INT &col, const INT &nnz, const std::vector< INT > &rowPtr, std::vector< INT > &colInd, std::vector< DBL > &values)

  Sort "colInd" of each row in ascending order and rearrange "values" accordingly.

10.15.1 Detailed Description

Some auxiliary functions for MAT.

Author

Chensong Zhang, Kailei Zhang

Date

Sep/26/2019

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

10.15.2 Function Documentation

10.15.2.1 CheckCSRx()

```
FaspRetCode CheckCSRx (
            const INT & row,
            const INT & col,
            const INT & nnz,
            const std::vector< DBL > & values,
            const std::vector< INT > & colInd,
            const std::vector< INT > & rowPtr,
            const std::vector< INT > & diagPtr )
```

Check whether the data is good for CSRx.
basic examinations
simple examinations
exam diagPtr and colInd

```
179 {
180     if ( row == 0 || col == 0 || nnz == 0 ) return FaspRetCode::SUCCESS;
181
182     /*
183      * some simple examinations about parameters
184      * to judge whether they are CSRx' parameters
185      */
186     /*----------------  begin  ----------------*/
188     INT count = 0;
189     INT begin, end;
190
191     if ( row != rowPtr.size() - 1 ) goto Return;
192
193     if ( row <= 0 || col <= 0 ) goto Return;
194
195     if (((row > col) ? col : row) != diagPtr.size()) goto Return;
196
197     if ( nnz != colInd.size()) goto Return;
198
199     if ( nnz != values.size()) goto Return;
200
201     if ( nnz != rowPtr[rowPtr.size() - 1] ) goto Return;
202
204     for ( INT j = 0; j < row; ++j ) {
```

```
205         if ( rowPtr[j] >= rowPtr[j + 1] ) goto Return;
206     }
207
208     if ( rowPtr[0] < 0 || rowPtr[row] > nnz ) goto Return;
209
210     for ( INT j = 0; j < row; ++j ) {
211         begin = rowPtr[j];
212         end = rowPtr[j + 1];
213         if ( begin == end ) goto Return;
214
215         if ( end == begin + 1 ) {
216             if ( colInd[begin] != j ) goto Return;
217         }
218
219         if ( end > begin + 1 ) {
220             for ( INT k = begin; k < end - 1; ++k ) {
221                 if ( colInd[k] >= colInd[k + 1] ) goto Return;
222             }
223             if ( 0 > colInd[begin] ) goto Return;
224
225             if ( colInd[end - 1] >= col ) goto Return;
226         }
227     }
228
230     for ( INT j = 0; j < row; ++j ) {
231         begin = rowPtr[j];
232         end = rowPtr[j + 1];
233         for ( INT k = begin; k < end; ++k ) {
234             if ( colInd[k] == j ) {
235                 if ( diagPtr[count] != k )
236                     goto Return;
237                 else
238                     ++count;
239             }
240         }
241     }
242     if ( count != diagPtr.size()) goto Return;
243
244     return FaspRetCode::SUCCESS;
245
246     Return: return FaspRetCode::ERROR_INPUT_PAR;
247 }
```

References ERROR_INPUT_PAR, and SUCCESS.

10.15.2.2 MTXtoMAT()

```
FaspRetCode MTXtoMAT (
              const INT & row,
              const INT & col,
              const INT & nnz,
              const std::vector< INT > & rowInd,
              const std::vector< INT > & colInd,
              const std::vector< DBL > & values,
              MAT & mat )
```

Convert MTX data to MAT.

Convert MTX data to MAT data structure.

```
420 {
421     auto retCode = FaspRetCode::SUCCESS;
422
423     std::vector<INT> rowPtrCSR;
424     std::vector<INT> colIndCSR;
425     std::vector<DBL> valuesCSR;
426
427     // Convert data format from MTX to CSR
428     MTXtoCSR(row, col, nnz, rowInd, colInd, values,valuesCSR, colIndCSR,rowPtrCSR);
429
430     // Sort CSR matrix row by row
431     SortCSRRow(row, col, nnz, rowPtrCSR, colIndCSR, valuesCSR);
432
433     // Check whether diagonal is a nonzero position
434     CSRtoMAT(row, col, nnz, valuesCSR, colIndCSR, rowPtrCSR, mat);
435
436     return retCode;
437 }
```

References SUCCESS.

10.16 MATUtil.hxx File Reference

Tools for checking and manipulating MAT.
`#include "MAT.hxx"`

Macros

- #define __MATUTIL_HXX__

Functions

- FaspRetCode CheckMATAddSize (const MAT &mat1, const MAT &mat2)

  Check whether two matrices have same sizes for addition.
- FaspRetCode CheckMATMultSize (const MAT &mat1, const MAT &mat2)

  Check MAT-MAT multiplication sizes.
- FaspRetCode CheckMATSize (const MAT &mat, const INT &row, const INT &col)

  Check whether (row,col) is out of bound.
- FaspRetCode CheckMATRowSize (const MAT &mat, const INT &row)

  Check whether (row,:) is out of bound.
- FaspRetCode CheckMATColSize (const MAT &mat, const INT &col)

  Check whether (:,col) is out of bound.
- FaspRetCode CheckMATVECSize (const MAT &mat, const VEC &vec)

  Check MAT-VEC multiplication sizes.
- FaspRetCode CheckCSR (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr)

  Check whether the data is good for CSR.
- FaspRetCode CheckCSRx (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, const std::vector< INT > &diagPtr)

  Check whether the data is good for CSRx.
- FaspRetCode CSRtoMAT (const INT &row, const INT &col, const INT &nnz, const std::vector< DBL > &values, const std::vector< INT > &colInd, const std::vector< INT > &rowPtr, MAT &mat)

  Convert a CSR matrix to MAT (private)
- FaspRetCode MTXtoMAT (const INT &row, const INT &col, const INT &nnz, const std::vector< INT > &rowInd, const std::vector< INT > &colInd, const std::vector< DBL > &values, MAT &mat)

  Convert MTX data to MAT data structure.
- FaspRetCode SortCSRRow (const INT &row, const INT &col, const INT &nnz, const std::vector< INT > &rowPtr, std::vector< INT > &colInd, std::vector< DBL > &values)

  Sort "colInd" of each row in ascending order and rearrange "values" accordingly.

10.16.1 Detailed Description

Tools for checking and manipulating MAT.

Author

Chensong Zhang, Kailei Zhang

Date

Sep/26/2019

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

10.16.2 Macro Definition Documentation

10.16.2.1 __MATUTIL_HXX__

```
#define __MATUTIL_HXX__
```

indicate MATUtil.hxx has been included before

10.16.3 Function Documentation

10.16.3.1 CheckCSRx()

```
FaspRetCode CheckCSRx (
             const INT & row,
             const INT & col,
             const INT & nnz,
             const std::vector< DBL > & values,
             const std::vector< INT > & colInd,
             const std::vector< INT > & rowPtr,
             const std::vector< INT > & diagPtr )
```

Check whether the data is good for CSRx.
basic examinations
simple examinations
exam diagPtr and colInd

```
179 {
180     if ( row == 0 || col == 0 || nnz == 0 ) return FaspRetCode::SUCCESS;
181
182     /*
183      * some simple examinations about parameters
184      * to judge whether they are CSRx' parameters
185      */
186     /*----------------  begin  ----------------*/
188     INT count = 0;
189     INT begin, end;
190
191     if ( row != rowPtr.size() - 1 ) goto Return;
192
193     if ( row <= 0 || col <= 0 ) goto Return;
194
195     if (((row > col) ? col : row) != diagPtr.size()) goto Return;
196
197     if ( nnz != colInd.size()) goto Return;
198
199     if ( nnz != values.size()) goto Return;
200
201     if ( nnz != rowPtr[rowPtr.size() - 1] ) goto Return;
202
204     for ( INT j = 0; j < row; ++j ) {
205         if ( rowPtr[j] >= rowPtr[j + 1] ) goto Return;
206     }
207
208     if ( rowPtr[0] < 0 || rowPtr[row] > nnz ) goto Return;
209
210     for ( INT j = 0; j < row; ++j ) {
211         begin = rowPtr[j];
212         end = rowPtr[j + 1];
213         if ( begin == end ) goto Return;
214
215         if ( end == begin + 1 ) {
216             if ( colInd[begin] != j ) goto Return;
217         }
218
219         if ( end > begin + 1 ) {
220             for ( INT k = begin; k < end - 1; ++k ) {
221                 if ( colInd[k] >= colInd[k + 1] ) goto Return;
222             }
223             if ( 0 > colInd[begin] ) goto Return;
224
225             if ( colInd[end - 1] >= col ) goto Return;
226         }
```

```
227     }
228
230     for ( INT j = 0; j < row; ++j ) {
231         begin = rowPtr[j];
232         end = rowPtr[j + 1];
233         for ( INT k = begin; k < end; ++k ) {
234             if ( colInd[k] == j ) {
235                 if ( diagPtr[count] != k )
236                     goto Return;
237                 else
238                     ++count;
239             }
240         }
241     }
242     if ( count != diagPtr.size()) goto Return;
243
244     return FaspRetCode::SUCCESS;
245
246     Return: return FaspRetCode::ERROR_INPUT_PAR;
247 }
```

References ERROR_INPUT_PAR, and SUCCESS.

10.16.3.2 MTXtoMAT()

```
FaspRetCode MTXtoMAT (
              const INT & row,
              const INT & col,
              const INT & nnz,
              const std::vector< INT > & rowInd,
              const std::vector< INT > & colInd,
              const std::vector< DBL > & values,
              MAT & mat )
```

Convert MTX data to MAT data structure.

Convert MTX data to MAT data structure.

```
420 {
421     auto retCode = FaspRetCode::SUCCESS;
422
423     std::vector<INT> rowPtrCSR;
424     std::vector<INT> colIndCSR;
425     std::vector<DBL> valuesCSR;
426
427     // Convert data format from MTX to CSR
428     MTXtoCSR(row, col, nnz, rowInd, colInd, values,valuesCSR, colIndCSR,rowPtrCSR);
429
430     // Sort CSR matrix row by row
431     SortCSRRow(row, col, nnz, rowPtrCSR, colIndCSR, valuesCSR);
432
433     // Check whether diagonal is a nonzero position
434     CSRtoMAT(row, col, nnz, valuesCSR, colIndCSR, rowPtrCSR, mat);
435
436     return retCode;
437 }
```

References SUCCESS.

10.17 Param.cxx File Reference

Command line input parameter definition.

```
#include <string>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <algorithm>
#include "Param.hxx"
#include "ErrorLog.hxx"
```

10.17.1 Detailed Description

Command line input parameter definition.

Author

Ronghong Fan, Chensong Zhang

Date

Nov/25/2019

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

10.18 Param.hxx File Reference

Command line input parameter declaration.

```
#include <utility>
#include <vector>
#include <string>
#include <iostream>
#include <cstring>
#include <fstream>
#include <map>
#include "faspxx.hxx"
```

Classes

- class Parameters

  Solver parameters.

Macros

- #define __PARAM_HEADER__

Enumerations

- enum Output {
  PRINT_NONE = 0, **PRINT_MIN** = 2, **PRINT_SOME** = 4, **PRINT_MORE** = 6,
  PRINT_MAX = 8 }

  Level of output.

10.18.1 Detailed Description

Command line input parameter declaration.

Author

Ronghong Fan, Chensong Zhang

Date

Nov/25/2019

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

10.18.2 Macro Definition Documentation

10.18.2.1 __PARAM_HEADER__

```
#define __PARAM_HEADER__
```
indicate Param.hxx has been included before

10.19 ReadData.cxx File Reference

Reading data from disk files.
```
#include <cstring>
#include <fstream>
#include "ReadData.hxx"
#include "MATUtil.hxx"
```

Functions

- FaspRetCode ReadVEC (const char *fileName, VEC &dst)

  Read a VEC data file stored as val[i], i=0:end-1.
- FaspRetCode ReadMTX (const char *fileName, INT &row, INT &col, INT &nnz, std::vector< INT > &rowInd, std::vector< INT > &colInd, std::vector< DBL > &values)

  Read (rowInd, colInd, values) from the MTX (MatrixMarket) file.
- FaspRetCode ReadCSR (const char *fileName, INT &row, INT &col, INT &nnz, std::vector< INT > &rowPtr, std::vector< INT > &colInd, std::vector< DBL > &values)

  Read (rowPtr, colInd, values) from the CSR file.
- FaspRetCode ReadMat (const char *fileName, MAT &dst)

  Read data from CSR or MTX file and store it in the MAT format.

10.19.1 Detailed Description

Reading data from disk files.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/11/2019

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

10.19.2 Function Documentation

10.19.2.1 ReadCSR()

```
FaspRetCode ReadCSR (
            const char * fileName,
            INT & row,
            INT & col,
            INT & nnz,
            std::vector< INT > & rowPtr,
            std::vector< INT > & colInd,
            std::vector< DBL > & values )
```
Read (rowPtr, colInd, values) from the CSR file.
Read a CSR data file and store it in (rowPtr, colInd, values)

```
202 {
203     FaspRetCode retCode = FaspRetCode::SUCCESS;
204 
205     // Open the file to read
206     std::cout « "Reading from disk file " « fileName « std::endl;
207     std::ifstream in(fileName);
208     if (!in.is_open()) { // judge whether file is opened successfully
209         std::cout « "Reading from disk file " « fileName « std::endl;
210         retCode = FaspRetCode::ERROR_OPEN_FILE;
211         return retCode;
212     }
213 
214     // Read the file in to a buffer
215     in.seekg(0, std::ios::end);
216     long long int length = in.tellg(); // compute total bytes 's number
217     in.seekg(0, std::ios::beg);
218 
219     char decimal[128];
220     char *buffer, *next;
221 
222     // Allocate memory space for storing the whole file
223     try { // catch the bad allocation if it happens
224         buffer = new char[length];
225     } catch (std::bad_alloc &ex) {
226         in.close();
227         retCode = FaspRetCode::ERROR_ALLOC_MEM;
228         return retCode;
229     }
230     in.read(buffer, length); // read the whole file in bytes
231     in.close(); // close the file stream
232 
233     // Read number of rows
234     INT count = 0;
235     long long int position = 0; // mark the position of file pointer
236     while (true) {
237         if (buffer[position] != '\n') {
238             decimal[count] = buffer[position];
239             ++count;
240             ++position;
241         } else {
242             decimal[count] = '\0'; // mark the end of 'decimal' string
243             ++position;
244             break;
245         }
246     }
247 
248     row = std::strtol(decimal, &next, 10);
249     if ( row <= 0 ) { // prevent memory leaks if error happens
250         retCode = FaspRetCode::ERROR_INPUT_PAR;
251         delete[] buffer;
252         return retCode;
253     }
254     col = row;
255 
256     // Read row pointers
257     try { // catch bad allocation if it happens
258         rowPtr.resize(row + 1);
259     } catch (std::bad_alloc &ex) {
260         retCode = FaspRetCode::ERROR_ALLOC_MEM;
261         return retCode;
262     }
263 
264     // Read the rowPtr of CSRx matrix
265     long int locate = 0;
266     count = 0;
267     while ( true ) {
268         if (buffer[position] != '\n') {
269             decimal[count] = buffer[position];
270             ++count;
271             ++position;
272         } else {
273             ++position;
274             decimal[count] = '\0';
275             count = 0;
276             rowPtr[locate] = std::strtol(decimal, &next, 10);
277             ++locate;
278             if (locate == row + 1) break;
279         }
280     }
281 
282     // Allocate memory for colInd and values
283     try { // catch bad allocation if it happens
284         nnz = rowPtr[row] - rowPtr[0];
285         colInd.resize(nnz);
286         values.resize(nnz);
287     } catch (std::bad_alloc &ex) {
288         retCode = FaspRetCode::ERROR_ALLOC_MEM;
```

```
289         return retCode;
290     }
291
292     // Read column indices
293     locate = 0;
294     while ( true ) {
295         if (buffer[position] != '\n') {
296             decimal[count] = buffer[position];
297             ++count;
298             ++position;
299         } else {
300             ++position;
301             decimal[count] = '\0';
302             count = 0;
303             colInd[locate] = std::strtol(decimal, &next, 10);
304             ++locate;
305             if (locate == nnz) break;
306         }
307     }
308
309     // Read values
310     locate = 0;
311     while ( true ) {
312         if (buffer[position] != '\n' && buffer[position] != '\0') {
313             decimal[count] = buffer[position];
314             ++count;
315             ++position;
316         } else {
317             if (buffer[position] == '\0') break;
318             ++position;
319             decimal[count] = '\0';
320             count = 0;
321             values[locate] = std::strtod(decimal, &next);
322             ++locate;
323         }
324     }
325     if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;
326
327     // If the indices start from 1, we shift them to start from 0
328     if ( rowPtr[0] == 1 ) {
329         for (count = 0; count <= row; ++count) rowPtr[count]--;
330         for (count = 0; count < nnz; ++count) colInd[count]--;
331     }
332
333     delete[] buffer; // clean up memory space
334
335     return retCode;
336 }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ERROR_OPEN_FILE, and S↩ UCCESS.

10.19.2.2 ReadMat()

```
FaspRetCode ReadMat (
             const char * fileName,
             MAT & dst )
```

Read data from CSR or MTX file and store it in the MAT format.

Read a MAT data file and store it in MAT.

```
340 {
341     const int len = strlen(fileName);
342     FaspRetCode retCode = FaspRetCode::SUCCESS;
343
344     if ( len <= 4 ) {
345         retCode = FaspRetCode::ERROR_INPUT_FILE;
346         return retCode;
347     }
348
349     // Check the file extension
350     char fileExt[4];
351     for ( int i = 0; i < 3; ++i ) fileExt[i] = tolower(fileName[len - 3 + i]);
352     fileExt[3] = '\0';
353
354     int flag = 0; // Undefined file format
355     if ( strcmp(fileExt, "csr" ) == 0)
356         flag = 1; // CSR file
357     else if ( strcmp(fileExt, "mtx" ) == 0)
358         flag = 2; // MTX file
359
360     INT row, col, nnz;
361     std::vector<INT> rowPtr, colInd, rowInd;
362     std::vector<DBL> values;
```

```
363
364     switch ( flag ) {
365         case 1:
366             try {
367                 retCode = ReadCSR(fileName, row, col, nnz,
368                                   rowPtr, colInd, values);
369                 if ( retCode < 0 )
370                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
371             }
372             catch (FaspRunTime &ex) {
373                 ex.LogExcep();
374                 break;
375             }
376
377             // Sort each row in ascending order
378             try {
379                 retCode = SortCSRRow(row, col, nnz, rowPtr, colInd, values);
380                 if ( retCode < 0 )
381                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
382             }
383             catch (FaspRunTime &ex) {
384                 ex.LogExcep();
385                 break;
386             }
387
388             // Convert a MTX matrix to MAT
389             try {
390                 retCode = CSRtoMAT(row, col, nnz, values, colInd, rowPtr, dst);
391                 if ( retCode < 0 )
392                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
393             }
394             catch (FaspRunTime &ex) {
395                 ex.LogExcep();
396                 break;
397             }
398             break;
399
400         case 2:
401             try {
402                 retCode = ReadMTX(fileName, row, col, nnz, rowInd,
403                                   colInd, values);
404                 if ( retCode < 0 )
405                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
406             }
407             catch (FaspRunTime &ex) {
408                 ex.LogExcep();
409                 break;
410             }
411
412             // Sort each row in ascending order
413             try {
414                 retCode = MTXtoMAT(row, col, nnz, rowInd, colInd, values, dst);
415                 if ( retCode < 0 )
416                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
417             }
418             catch (FaspRunTime &ex) {
419                 ex.LogExcep();
420                 break;
421             }
422             break;
423
424         default:
425             FASPXX_WARNING("Unknown file format detected!")
426             retCode = FaspRetCode::ERROR_INPUT_FILE;
427             break;
428     }
429
430     return retCode;
431 }
```

References CSRtoMAT(), ERROR_INPUT_FILE, FASPXX_WARNING, FaspRunTime::LogExcep(), MTXtoMAT(), ReadCSR(), ReadMTX(), SortCSRRow(), and SUCCESS.

10.19.2.3 ReadMTX()

```
FaspRetCode ReadMTX (
            const char * fileName,
            INT & row,
            INT & col,
            INT & nnz,
            std::vector< INT > & rowInd,
```

```
            std::vector< INT > & colInd,
            std::vector< DBL > & values )
```

Read (rowInd, colInd, values) from the MTX (MatrixMarket) file.

Read an MTX data file and store it in (rowInd, colInd, values)

```
95 {
96     FaspRetCode retCode = FaspRetCode::SUCCESS;
97
98     // Open the file to read
99     std::cout « "Reading from disk file " « fileName « std::endl;
100     std::ifstream in(fileName);
101     if (!in.is_open()) { // check whether file is opened successfully
102         retCode = FaspRetCode::ERROR_OPEN_FILE;
103         return retCode;
104     }
105
106     // Read the file in to a buffer
107     in.seekg(0, std::ios::end);
108     const long long int length = in.tellg();
109     in.seekg(0, std::ios::beg);
110
111     char decimal[128];
112     char *buffer, *next;
113     long long int position = 0; // position of file pointer
114
115     // Allocate temp space for storing the whole file
116     try { // catch bad allocation if it happens
117         buffer = new char[length];
118     } catch (std::bad_alloc &ex) {
119         in.close();
120         retCode = FaspRetCode::ERROR_ALLOC_MEM;
121         return retCode;
122     }
123     in.read(buffer, length); // read the whole file in bytes
124     in.close(); // close the file stream
125
126     int count = 0; // number of bytes in the decimal
127     int mark = 0; // which number of integer is reading
128     while ( true ) { // read matrix 's row, column, nnz
129         if ( buffer[position] != ' ' && buffer[position] != '\n' ) {
130             decimal[count] = buffer[position];
131             ++count;
132             ++position;
133         } else {
134             decimal[count] = '\0';
135             count = 0;
136             ++mark;
137             ++position;
138             switch (mark) {
139                 case 1: // first, integer, number of rows
140                     row = std::strtol(decimal, &next, 10); break;
141                 case 2: // second, integer, number of columns
142                     col = std::strtol(decimal, &next, 10); break;
143                 case 3: // third, integer, number of nonzeros
144                     nnz = std::strtol(decimal, &next, 10); break;
145                 default:
146                     FASPXX_WARNING("Unknown input value!")
147             }
148         }
149         if ( mark == 3 ) break; // skip the rest
150     }
151
152     // Allocate memory space to store row indices, column indices and values
153     try { // catch the bad allocation if it happens
154         rowInd.resize(nnz);
155         colInd.resize(nnz);
156         values.resize(nnz);
157     } catch (std::bad_alloc &ex) {
158         delete[] buffer; // if bad allocation happens, free up the memory space
159         retCode = FaspRetCode::ERROR_ALLOC_MEM;
160         return retCode;
161     }
162
163     // Put MTX data into rowInd, colInd, and values
164     long int locate = 0; // mark the position in rowInd, colInd and values
165     long int tmp = 0;
166     while ( true ) {
167         if (buffer[position] != ' ' && buffer[position] != '\n' &&
168             buffer[position] != '\0') {
169             decimal[count] = buffer[position];
170             ++count;
171             ++position;
172         } else {
173             ++position;
174             if (buffer[position] == ' ') continue; // multiple consecutive spaces
175             decimal[count] = '\0'; // mark the end of 'decimal' string
176             count = 0;
```

```
177             ++tmp;
178             locate = tmp / 3;
179             switch (tmp % 3) {
180                 case 1: // first: integer, row index
181                     rowInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
182                 case 2: // second: integer, column index
183                     colInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
184                 case 0: // third: double, value
185                     values[locate-1] = std::strtod(decimal, &next); break;
186             }
187             if (buffer[position] == '\0') break;
188         }
189     }
190 
191     if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;
192 
193     delete[] buffer; // clean up memory space
194 
195     return retCode;
196 }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_OPEN_FILE, FASPXX_WARNING, and S↩UCCESS.

10.19.2.4 ReadVEC()

```
FaspRetCode ReadVEC (
             const char * fileName,
             VEC & dst )
```

Read a VEC data file stored as val[i], i=0:end-1.

Read a VEC data file and store it in dst.

```
19 {
20     FaspRetCode retCode = FaspRetCode::SUCCESS;
21 
22     std::cout « "Reading from disk file " « fileName « std::endl;
23     std::ifstream in(fileName);
24     if ( !in.is_open() ) { // check whether file is opened successfully
25         retCode = FaspRetCode::ERROR_OPEN_FILE;
26         return retCode;
27     }
28 
29     // Compute total number of bytes of file
30     in.seekg(0, std::ios::end);
31     const long long int length = in.tellg();
32     in.seekg(0, std::ios::beg);
33 
34     char decimal[128]; // temporary storage for data
35     long long int position = 0; // mark the position of file pointer
36     long int count = 0, len;
37 
38     char *buffer, *next;
39     try { // catch bad allocation error if it happens
40         buffer = new char[length]; // allocate memory for buffer
41     } catch (std::bad_alloc &ex) {
42         in.close();
43         retCode = FaspRetCode::ERROR_ALLOC_MEM;
44         return retCode;
45     }
46     in.read(buffer, length); // read the total bytes of file
47     in.close(); // close the file pointer
48 
49     // Read in the size of VEC object
50     while ( true ) {
51         if (buffer[position] != '\n') {
52             decimal[count] = buffer[position];
53             ++position;
54             ++count;
55         } else {
56             decimal[count] = '\0';
57             count = 0;
58             ++position;
59             len = std::strtol(decimal, &next, 10);
60             break;
61         }
62     }
63 
64     // Allocate memory space and initialize
65     dst.SetValues(len, 0.0);
66 
67     // Read in the VEC object's entries
68     long int locate = 0; // mark the element position
69     while ( true ) {
```

```
70          if (buffer[position] != '\n') {
71              decimal[count] = buffer[position];
72              ++position;
73              ++count;
74          } else {
75              decimal[count] = '\0';
76              count = 0;
77              ++position;
78              dst[locate] = std::strtod(decimal, &next);
79              ++locate;
80          }
81          if (buffer[position] == '\0') break;
82      }
83
84      if ( locate != len ) retCode = FaspRetCode::ERROR_INPUT_FILE;
85
86      delete[] buffer; // clean up memory space
87
88      return retCode;
89  }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_OPEN_FILE, VEC::SetValues(), and SUC↩ CESS.

10.20 ReadData.hxx File Reference

Reading data from disk files.

```
#include "faspxx.hxx"
#include "MAT.hxx"
```

Macros

- #define __READDATA__HEADER__

Functions

- FaspRetCode ReadVEC (const char *filename, VEC &dst)

  Read a VEC data file and store it in dst.
- FaspRetCode ReadMTX (const char *filename, INT &row, INT &col, INT &nnz, std::vector< INT > &rowInd, std::vector< INT > &colInd, std::vector< DBL > &values)

  Read an MTX data file and store it in (rowInd, colInd, values)
- FaspRetCode ReadCSR (const char *filename, INT &row, INT &col, INT &nnz, std::vector< INT > &rowPtr, std::vector< INT > &colInd, std::vector< DBL > &values)

  Read a CSR data file and store it in (rowPtr, colInd, values)
- FaspRetCode ReadMat (const char *filename, MAT &dst)

  Read a MAT data file and store it in MAT.

10.20.1 Detailed Description

Reading data from disk files.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/11/2019

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

10.20.2 Macro Definition Documentation

10.20.2.1 __READDATA__HEADER__

```
#define __READDATA__HEADER__
```

indicate ReadData.hxx has been included before

10.20.3 Function Documentation

10.20.3.1 ReadCSR()

```
FaspRetCode ReadCSR (
            const char * fileName,
            INT & row,
            INT & col,
            INT & nnz,
            std::vector< INT > & rowPtr,
            std::vector< INT > & colInd,
            std::vector< DBL > & values )
```

Read a CSR data file and store it in (rowPtr, colInd, values)

Read a CSR data file and store it in (rowPtr, colInd, values)

```
202 {
203     FaspRetCode retCode = FaspRetCode::SUCCESS;
204 
205     // Open the file to read
206     std::cout « "Reading from disk file " « fileName « std::endl;
207     std::ifstream in(fileName);
208     if (!in.is_open()) { // judge whether file is opened successfully
209         std::cout « "Reading from disk file " « fileName « std::endl;
210         retCode = FaspRetCode::ERROR_OPEN_FILE;
211         return retCode;
212     }
213 
214     // Read the file in to a buffer
215     in.seekg(0, std::ios::end);
216     long long int length = in.tellg(); // compute total bytes 's number
217     in.seekg(0, std::ios::beg);
218 
219     char decimal[128];
220     char *buffer, *next;
221 
222     // Allocate memory space for storing the whole file
223     try { // catch the bad allocation if it happens
224         buffer = new char[length];
225     } catch (std::bad_alloc &ex) {
226         in.close();
227         retCode = FaspRetCode::ERROR_ALLOC_MEM;
228         return retCode;
229     }
230     in.read(buffer, length); // read the whole file in bytes
231     in.close(); // close the file stream
232 
233     // Read number of rows
234     INT count = 0;
235     long long int position = 0; // mark the position of file pointer
236     while (true) {
237         if (buffer[position] != '\n') {
238             decimal[count] = buffer[position];
239             ++count;
240             ++position;
241         } else {
242             decimal[count] = '\0'; // mark the end of 'decimal' string
243             ++position;
244             break;
245         }
246     }
247 
248     row = std::strtol(decimal, &next, 10);
249     if ( row <= 0 ) { // prevent memory leaks if error happens
250         retCode = FaspRetCode::ERROR_INPUT_PAR;
251         delete[] buffer;
252         return retCode;
253     }
254     col = row;
255 
256     // Read row pointers
257     try { // catch bad allocation if it happens
258         rowPtr.resize(row + 1);
```

```
259     } catch (std::bad_alloc &ex) {
260         retCode = FaspRetCode::ERROR_ALLOC_MEM;
261         return retCode;
262     }
263 
264     // Read the rowPtr of CSRx matrix
265     long int locate = 0;
266     count = 0;
267     while ( true ) {
268         if (buffer[position] != '\n') {
269             decimal[count] = buffer[position];
270             ++count;
271             ++position;
272         } else {
273             ++position;
274             decimal[count] = '\0';
275             count = 0;
276             rowPtr[locate] = std::strtol(decimal, &next, 10);
277             ++locate;
278             if (locate == row + 1) break;
279         }
280     }
281 
282     // Allocate memory for colInd and values
283     try { // catch bad allocation if it happens
284         nnz = rowPtr[row] - rowPtr[0];
285         colInd.resize(nnz);
286         values.resize(nnz);
287     } catch (std::bad_alloc &ex) {
288         retCode = FaspRetCode::ERROR_ALLOC_MEM;
289         return retCode;
290     }
291 
292     // Read column indices
293     locate = 0;
294     while ( true ) {
295         if (buffer[position] != '\n') {
296             decimal[count] = buffer[position];
297             ++count;
298             ++position;
299         } else {
300             ++position;
301             decimal[count] = '\0';
302             count = 0;
303             colInd[locate] = std::strtol(decimal, &next, 10);
304             ++locate;
305             if (locate == nnz) break;
306         }
307     }
308 
309     // Read values
310     locate = 0;
311     while ( true ) {
312         if (buffer[position] != '\n' && buffer[position] != '\0') {
313             decimal[count] = buffer[position];
314             ++count;
315             ++position;
316         } else {
317             if (buffer[position] == '\0') break;
318             ++position;
319             decimal[count] = '\0';
320             count = 0;
321             values[locate] = std::strtod(decimal, &next);
322             ++locate;
323         }
324     }
325     if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;
326 
327     // If the indices start from 1, we shift them to start from 0
328     if ( rowPtr[0] == 1 ) {
329         for (count = 0; count <= row; ++count) rowPtr[count]--;
330         for (count = 0; count < nnz; ++count) colInd[count]--;
331     }
332 
333     delete[] buffer; // clean up memory space
334 
335     return retCode;
336 }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ERROR_OPEN_FILE, and S↩
UCCESS.

10.20.3.2 ReadMat()

```
FaspRetCode ReadMat (
            const char * fileName,
            MAT & dst )
```

Read a MAT data file and store it in MAT.

Read a MAT data file and store it in MAT.

```
340 {
341     const int len = strlen(fileName);
342     FaspRetCode retCode = FaspRetCode::SUCCESS;
343 
344     if ( len <= 4 ) {
345         retCode = FaspRetCode::ERROR_INPUT_FILE;
346         return retCode;
347     }
348 
349     // Check the file extension
350     char fileExt[4];
351     for ( int i = 0; i < 3; ++i ) fileExt[i] = tolower(fileName[len - 3 + i]);
352     fileExt[3] = '\0';
353 
354     int flag = 0; // Undefined file format
355     if ( strcmp(fileExt, "csr" ) == 0)
356         flag = 1; // CSR file
357     else if ( strcmp(fileExt, "mtx" ) == 0)
358         flag = 2; // MTX file
359 
360     INT row, col, nnz;
361     std::vector<INT> rowPtr, colInd, rowInd;
362     std::vector<DBL> values;
363 
364     switch ( flag ) {
365         case 1:
366             try {
367                 retCode = ReadCSR(fileName, row, col, nnz,
368                                   rowPtr, colInd, values);
369                 if ( retCode < 0 )
370                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
371             }
372             catch (FaspRunTime &ex) {
373                 ex.LogExcep();
374                 break;
375             }
376 
377             // Sort each row in ascending order
378             try {
379                 retCode = SortCSRRow(row, col, nnz, rowPtr, colInd, values);
380                 if ( retCode < 0 )
381                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
382             }
383             catch (FaspRunTime &ex) {
384                 ex.LogExcep();
385                 break;
386             }
387 
388             // Convert a MTX matrix to MAT
389             try {
390                 retCode = CSRtoMAT(row, col, nnz, values, colInd, rowPtr, dst);
391                 if ( retCode < 0 )
392                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
393             }
394             catch (FaspRunTime &ex) {
395                 ex.LogExcep();
396                 break;
397             }
398             break;
399 
400         case 2:
401             try {
402                 retCode = ReadMTX(fileName, row, col, nnz, rowInd,
403                                   colInd, values);
404                 if ( retCode < 0 )
405                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
406             }
407             catch (FaspRunTime &ex) {
408                 ex.LogExcep();
409                 break;
410             }
411 
412             // Sort each row in ascending order
413             try {
414                 retCode = MTXtoMAT(row, col, nnz, rowInd, colInd, values, dst);
415                 if ( retCode < 0 )
416                     throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
417             }
```

```
418             catch (FaspRunTime &ex) {
419                 ex.LogExcep();
420                 break;
421             }
422             break;
423 
424         default:
425             FASPXX_WARNING("Unknown file format detected!")
426             retCode = FaspRetCode::ERROR_INPUT_FILE;
427             break;
428     }
429 
430     return retCode;
431 }
```

References CSRtoMAT(), ERROR_INPUT_FILE, FASPXX_WARNING, FaspRunTime::LogExcep(), MTXtoMAT(), ReadCSR(), ReadMTX(), SortCSRRow(), and SUCCESS.

10.20.3.3 ReadMTX()

```
FaspRetCode ReadMTX (
              const char * fileName,
              INT & row,
              INT & col,
              INT & nnz,
              std::vector< INT > & rowInd,
              std::vector< INT > & colInd,
              std::vector< DBL > & values )
```

Read an MTX data file and store it in (rowInd, colInd, values)

Read an MTX data file and store it in (rowInd, colInd, values)

```
95 {
96     FaspRetCode retCode = FaspRetCode::SUCCESS;
97 
98     // Open the file to read
99     std::cout « "Reading from disk file " « fileName « std::endl;
100     std::ifstream in(fileName);
101     if (!in.is_open()) { // check whether file is opened successfully
102         retCode = FaspRetCode::ERROR_OPEN_FILE;
103         return retCode;
104     }
105 
106     // Read the file in to a buffer
107     in.seekg(0, std::ios::end);
108     const long long int length = in.tellg();
109     in.seekg(0, std::ios::beg);
110 
111     char decimal[128];
112     char *buffer, *next;
113     long long int position = 0; // position of file pointer
114 
115     // Allocate temp space for storing the whole file
116     try { // catch bad allocation if it happens
117         buffer = new char[length];
118     } catch (std::bad_alloc &ex) {
119         in.close();
120         retCode = FaspRetCode::ERROR_ALLOC_MEM;
121         return retCode;
122     }
123     in.read(buffer, length); // read the whole file in bytes
124     in.close(); // close the file stream
125 
126     int count = 0; // number of bytes in the decimal
127     int mark = 0; // which number of integer is reading
128     while ( true ) { // read matrix 's row, column, nnz
129         if ( buffer[position] != ' ' && buffer[position] != '\n' ) {
130             decimal[count] = buffer[position];
131             ++count;
132             ++position;
133         } else {
134             decimal[count] = '\0';
135             count = 0;
136             ++mark;
137             ++position;
138             switch (mark) {
139                 case 1: // first, integer, number of rows
140                     row = std::strtol(decimal, &next, 10); break;
141                 case 2: // second, integer, number of columns
142                     col = std::strtol(decimal, &next, 10); break;
143                 case 3: // third, integer, number of nonzeros
```

```
144                     nnz = std::strtol(decimal, &next, 10); break;
145                 default:
146                     FASPXX_WARNING("Unknown input value!")
147             }
148         }
149         if ( mark == 3 ) break; // skip the rest
150     }
151 
152     // Allocate memory space to store row indices, column indices and values
153     try { // catch the bad allocation if it happens
154         rowInd.resize(nnz);
155         colInd.resize(nnz);
156         values.resize(nnz);
157     } catch (std::bad_alloc &ex) {
158         delete[] buffer; // if bad allocation happens, free up the memory space
159         retCode = FaspRetCode::ERROR_ALLOC_MEM;
160         return retCode;
161     }
162 
163     // Put MTX data into rowInd, colInd, and values
164     long int locate = 0; // mark the position in rowInd, colInd and values
165     long int tmp = 0;
166     while ( true ) {
167         if (buffer[position] != ' ' && buffer[position] != '\n' &&
168             buffer[position] != '\0') {
169             decimal[count] = buffer[position];
170             ++count;
171             ++position;
172         } else {
173             ++position;
174             if (buffer[position] == ' ') continue; // multiple consecutive spaces
175             decimal[count] = '\0'; // mark the end of 'decimal' string
176             count = 0;
177             ++tmp;
178             locate = tmp / 3;
179             switch (tmp % 3) {
180                 case 1: // first: integer, row index
181                     rowInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
182                 case 2: // second: integer, column index
183                     colInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
184                 case 0: // third: double, value
185                     values[locate-1] = std::strtod(decimal, &next); break;
186             }
187             if (buffer[position] == '\0') break;
188         }
189     }
190 
191     if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;
192 
193     delete[] buffer; // clean up memory space
194 
195     return retCode;
196 }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_OPEN_FILE, FASPXX_WARNING, and S↩ UCCESS.

10.20.3.4 ReadVEC()

```
FaspRetCode ReadVEC (
              const char * fileName,
              VEC & dst )
```

Read a VEC data file and store it in dst.

Read a VEC data file and store it in dst.

```
19 {
20     FaspRetCode retCode = FaspRetCode::SUCCESS;
21 
22     std::cout « "Reading from disk file " « fileName « std::endl;
23     std::ifstream in(fileName);
24     if ( !in.is_open() ) { // check whether file is opened successfully
25         retCode = FaspRetCode::ERROR_OPEN_FILE;
26         return retCode;
27     }
28 
29     // Compute total number of bytes of file
30     in.seekg(0, std::ios::end);
31     const long long int length = in.tellg();
32     in.seekg(0, std::ios::beg);
33 
34     char decimal[128]; // temporary storage for data
35     long long int position = 0; // mark the position of file pointer
36     long int count = 0, len;
```

```
37
38      char *buffer, *next;
39      try { // catch bad allocation error if it happens
40          buffer = new char[length]; // allocate memory for buffer
41      } catch (std::bad_alloc &ex) {
42          in.close();
43          retCode = FaspRetCode::ERROR_ALLOC_MEM;
44          return retCode;
45      }
46      in.read(buffer, length); // read the total bytes of file
47      in.close(); // close the file pointer
48
49      // Read in the size of VEC object
50      while ( true ) {
51          if (buffer[position] != '\n') {
52              decimal[count] = buffer[position];
53              ++position;
54              ++count;
55          } else {
56              decimal[count] = '\0';
57              count = 0;
58              ++position;
59              len = std::strtol(decimal, &next, 10);
60              break;
61          }
62      }
63
64      // Allocate memory space and initialize
65      dst.SetValues(len, 0.0);
66
67      // Read in the VEC object's entries
68      long int locate = 0; // mark the element position
69      while ( true ) {
70          if (buffer[position] != '\n') {
71              decimal[count] = buffer[position];
72              ++position;
73              ++count;
74          } else {
75              decimal[count] = '\0';
76              count = 0;
77              ++position;
78              dst[locate] = std::strtod(decimal, &next);
79              ++locate;
80          }
81          if (buffer[position] == '\0') break;
82      }
83
84      if ( locate != len ) retCode = FaspRetCode::ERROR_INPUT_FILE;
85
86      delete[] buffer; // clean up memory space
87
88      return retCode;
89  }
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_OPEN_FILE, VEC::SetValues(), and SUCCESS.

10.21 RetCode.hxx File Reference

Decode return code into a readable string.

```
#include <string>
#include <ostream>
#include <iostream>
```

Classes

- class FaspRunTime

  Run-time exception capturing class.

- class FaspBadAlloc

  Allocation exception capturing class.

Macros

- #define __RETCODE_HEADER__

Enumerations

- enum FaspRetCode {
  SUCCESS = 0, ERROR_OPEN_FILE = -10, ERROR_INPUT_FILE = -11, ERROR_INPUT_PAR = -12,
  ERROR_VEC_SIZE = -14, ERROR_MAT_SIZE = -15, ERROR_NONMATCH_SIZE = -16, ERROR_MAT_DATA = -17,
  ERROR_DIVIDE_ZERO = -18, ERROR_MAT_ZERODIAG = -19, ERROR_ALLOC_MEM = -20, ERROR_DUMMY_VAR = -23,
  ERROR_SOLVER_TYPE = -30, ERROR_SOLVER_PRECTYPE = -31, ERROR_SOLVER_STAG = -32, ERROR_SOLVER_SOLSTAG = -33,
  ERROR_SOLVER_TOLSMALL = -34, ERROR_SOLVER_MAXIT = -39, ERROR_AMG_INTERP_TYPE = -40, ERROR_AMG_SMOOTH_TYPE = -41,
  ERROR_AMG_COARSE_TYPE = -42, ERROR_AMG_COARSEING = -43, ERROR_AMG_SETUP = -49, ERROR_ILU_TYPE = -50,
  ERROR_ILU_SETUP = -59, ERROR_SWZ_TYPE = -60, ERROR_SWZ_SETUP = -69, ERROR_UNKNOWN = -99 }

  Return code definition.

Functions

- std::string GetRetCode (const FaspRetCode code)

  Get error message from FaspRetCode.

10.21.1 Detailed Description

Decode return code into a readable string.
Exception types and return code definitions.

Author

Chensong Zhang

Date

Sep/25/2019

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

Author

Chensong Zhang

Date

Sep/12/2019

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

10.21.2 Macro Definition Documentation

10.21.2.1 __RETCODE_HEADER__

```
#define __RETCODE_HEADER__
```

indicate RetCode.hxx has been included before

10.21.3 Enumeration Type Documentation

10.21.3.1 FaspRetCode

enum FaspRetCode
Return code definition.

Enumerator

SUCCESS	Everything is fine.
ERROR_OPEN_FILE	Failed to open a file.
ERROR_INPUT_FILE	Wrong input file.
ERROR_INPUT_PAR	Wrong input argument.
ERROR_VEC_SIZE	Wrong vector size.
ERROR_MAT_SIZE	Wrong matrix size.
ERROR_NONMATCH_SIZE	Two sizes do not match.
ERROR_MAT_DATA	Wrong matrix format.
ERROR_DIVIDE_ZERO	Divided by zero!
ERROR_MAT_ZERODIAG	MAT has zero diagonal entries.
ERROR_ALLOC_MEM	Failed to allocate memory.
ERROR_DUMMY_VAR	Unknown function dummy variables.
ERROR_SOLVER_TYPE	Unknown solver type.
ERROR_SOLVER_PRECTYPE	Unknown preconditioner type.
ERROR_SOLVER_STAG	Iterative solver stagnates.
ERROR_SOLVER_SOLSTAG	Iterative solver's solution is too small.
ERROR_SOLVER_TOLSMALL	Iterative solver's tolerance is too small.
ERROR_SOLVER_MAXIT	Maximal iteration number reached.
ERROR_AMG_INTERP_TYPE	Unknown AMG interpolation type.
ERROR_AMG_SMOOTH_TYPE	Unknown AMG smoother type.
ERROR_AMG_COARSE_TYPE	Unknown AMG coarsening type.
ERROR_AMG_COARSEING	AMG coarsening step failed to complete.
ERROR_AMG_SETUP	AMG setup failed to complete.
ERROR_ILU_TYPE	Unknown ILU method type.
ERROR_ILU_SETUP	ILU setup failed to complete.
ERROR_SWZ_TYPE	Unknown Schwarz method type.
ERROR_SWZ_SETUP	Schwarz method setup failed to complete.
ERROR_UNKNOWN	Unknown error type.

```
21 {
22     SUCCESS               = 0,
23     //---------- Input problems ----------------------------------------------------//
24     ERROR_OPEN_FILE       = -10,
25     ERROR_INPUT_FILE      = -11,
26     ERROR_INPUT_PAR       = -12,
27     //---------- VEC or MAT data problems ------------------------------------------//
28     ERROR_VEC_SIZE        = -14,
29     ERROR_MAT_SIZE        = -15,
30     ERROR_NONMATCH_SIZE   = -16,
31     ERROR_MAT_DATA        = -17,
32     ERROR_DIVIDE_ZERO     = -18,
33     ERROR_MAT_ZERODIAG    = -19,
34     //---------- Memory or function call problems ----------------------------------//
35     ERROR_ALLOC_MEM       = -20,
36     ERROR_DUMMY_VAR       = -23,
37     //---------- Iterative method problems -----------------------------------------//
38     ERROR_SOLVER_TYPE     = -30,
39     ERROR_SOLVER_PRECTYPE = -31,
40     ERROR_SOLVER_STAG     = -32,
```

```
41     ERROR_SOLVER_SOLSTAG   = -33,
42     ERROR_SOLVER_TOLSMALL  = -34,
43     ERROR_SOLVER_MAXIT     = -39,
44     //---------- AMG method problems ----------------------------------------------//
45     ERROR_AMG_INTERP_TYPE  = -40,
46     ERROR_AMG_SMOOTH_TYPE  = -41,
47     ERROR_AMG_COARSE_TYPE  = -42,
48     ERROR_AMG_COARSEING    = -43,
49     ERROR_AMG_SETUP        = -49,
50     //---------- ILU method problems ----------------------------------------------//
51     ERROR_ILU_TYPE         = -50,
52     ERROR_ILU_SETUP        = -59,
53     //---------- ILU method problems ----------------------------------------------//
54     ERROR_SWZ_TYPE         = -60,
55     ERROR_SWZ_SETUP        = -69,
56     //---------- Unknown problems (default) ---------------------------------------//
57     ERROR_UNKNOWN          = -99,
58 };
```

10.21.4 Function Documentation

10.21.4.1 GetRetCode()

```
std::string GetRetCode (
            const FaspRetCode code )
```

Get error message from FaspRetCode.

Get error message from FaspRetCode.

```
16                                          {
17     switch ( code ) {
18         case SUCCESS:
19             return "Finish successfully!";
20         case ERROR_OPEN_FILE:
21             return "Failed to open a file!";
22         case ERROR_INPUT_FILE:
23             return "Wrong input file!";
24         case ERROR_INPUT_PAR:
25             return "Wrong input argument!";
26         case ERROR_VEC_SIZE:
27             return "Wrong vector size!";
28         case ERROR_MAT_SIZE:
29             return "Wrong matrix size!";
30         case ERROR_NONMATCH_SIZE:
31             return "Two sizes do not match!";
32         case ERROR_MAT_DATA:
33             return "Wrong matrix format!";
34         case ERROR_DIVIDE_ZERO:
35             return "Divided by zero!";
36         case ERROR_MAT_ZERODIAG:
37             return "MAT has zero diagonal entries!";
38         case ERROR_ALLOC_MEM:
39             return "Failed to allocate memory!";
40         case ERROR_DUMMY_VAR:
41             return "Unknown function dummy variables!";
42         case ERROR_SOLVER_TYPE:
43             return "Unknown solver type!";
44         case ERROR_SOLVER_PRECTYPE:
45             return "Unknown preconditioner type!";
46         case ERROR_SOLVER_STAG:
47             return "Iterative solver stagnates!";
48         case ERROR_SOLVER_SOLSTAG:
49             return "Iterative solver's solution is too small!";
50         case ERROR_SOLVER_TOLSMALL:
51             return "Iterative solver's tolerance is too small!";
52         case ERROR_SOLVER_MAXIT:
53             return "Maximal iteration number reached!";
54         case ERROR_AMG_INTERP_TYPE:
55             return "Unknown AMG interpolation type!";
56         case ERROR_AMG_SMOOTH_TYPE:
57             return "Unknown AMG smoother type!";
58         case ERROR_AMG_COARSE_TYPE:
59             return "Unknown AMG coarsening type!";
60         case ERROR_AMG_COARSEING:
61             return "AMG coarsening step failed to complete!";
62         case ERROR_AMG_SETUP:
63             return "AMG setup failed to complete!";
64         case ERROR_ILU_TYPE:
65             return "Unknown ILU method type";
66         case ERROR_ILU_SETUP:
67             return "ILU setup failed to complete!";
68         case ERROR_SWZ_TYPE:
```

```
69             return "Unknown Schwarz method type";
70         case ERROR_SWZ_SETUP:
71             return "Schwarz method setup failed to complete!";
72         default:
73             return "Unknown error type!";
74     }
75 }
```

References ERROR_ALLOC_MEM, ERROR_AMG_COARSE_TYPE, ERROR_AMG_COARSEING, ERROR_A↩MG_INTERP_TYPE, ERROR_AMG_SETUP, ERROR_AMG_SMOOTH_TYPE, ERROR_DIVIDE_ZERO, ERRO↩R_DUMMY_VAR, ERROR_ILU_SETUP, ERROR_ILU_TYPE, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ER↩ROR_MAT_DATA, ERROR_MAT_SIZE, ERROR_MAT_ZERODIAG, ERROR_NONMATCH_SIZE, ERROR_OP↩EN_FILE, ERROR_SOLVER_MAXIT, ERROR_SOLVER_PRECTYPE, ERROR_SOLVER_SOLSTAG, ERROR↩_SOLVER_STAG, ERROR_SOLVER_TOLSMALL, ERROR_SOLVER_TYPE, ERROR_SWZ_SETUP, ERROR↩_SWZ_TYPE, ERROR_VEC_SIZE, and SUCCESS.

10.22 SOL.cxx File Reference

Iterative solver class definition.

```
#include <sstream>
#include "SOL.hxx"
```

10.22.1 Detailed Description

Iterative solver class definition.

Author

Kailei Zhang, Chensong Zhang

Date

Nov/25/2019

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

10.23 SOL.hxx File Reference

Iterative solver class declaration.

```
#include <cstring>
#include <iomanip>
#include <iostream>
#include <fstream>
#include "faspxx.hxx"
#include "RetCode.hxx"
#include "ErrorLog.hxx"
#include "Param.hxx"
#include "LOP.hxx"
#include "VEC.hxx"
```

Classes

- struct SOLParams

  Iterative solver parameters.

- class SOL

  Base class for iterative solvers.

Macros

- #define __SOL_HEADER__

Enumerations

- enum SOLType {
  CG = 1, BICGSTAB = 2, MINRES = 3, GMRES = 4,
  FGMRES = 5, VFGMRES = 6, Jacobi = 11 }

  Iterative solver type.

10.23.1 Detailed Description

Iterative solver class declaration.

Author

Kailei Zhang, Chensong Zhang, Ronghong Fan

Date

Nov/25/2019

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

10.23.2 Macro Definition Documentation

10.23.2.1 __SOL_HEADER__

```
#define __SOL_HEADER__
```

indicate SOL.hxx has been included before

10.23.3 Enumeration Type Documentation

10.23.3.1 SOLType

```
enum SOLType
```

Iterative solver type.

Enumerator

CG	Conjugate Gradient.
BICGSTAB	Bi-Conjugate Gradient Stabilized.
MINRES	Minimal Residual.
GMRES	Generalized Minimal Residual.
FGMRES	Flexible GMRES.
VFGMRES	Variable-restarting FGMRES.
Jacobi	Jacobi iteration.

```
29             {
30     CG        = 1,
31     BICGSTAB  = 2,
32     MINRES    = 3,
33     GMRES     = 4,
34     FGMRES    = 5,
35     VFGMRES   = 6,
```

```
36     Jacobi    = 11,
37 };
```

10.24 Timing.hxx File Reference

Measure elapsed wall-time and CPU-cycles.
`#include <chrono>`

Classes

- class GetWallTime

  Get elapsed wall-time in millisecond.
- class GetCycleNum

  Get CPU-cycle number.

Macros

- #define __TIMING_HEADER__

Typedefs

- typedef unsigned long long uint64

  Unsigned long long int.

10.24.1 Detailed Description

Measure elapsed wall-time and CPU-cycles.

Author

Chensong Zhang

Date

Sep/24/2019

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

10.24.2 Macro Definition Documentation

10.24.2.1 __TIMING_HEADER__

`#define __TIMING_HEADER__`
indicate timing.hxx has been included –

10.25 VEC.cxx File Reference

Vector class definition.
`#include <cmath>`
`#include "VEC.hxx"`

10.25.1 Detailed Description

Vector class definition.

Author

Chensong Zhang, Kailei Zhang

Date

Oct/13/2019

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

10.26 VEC.hxx File Reference

Vector class declaration.

```
#include <vector>
#include "faspxx.hxx"
#include "RetCode.hxx"
```

Classes

- class VEC

  General vector class.

Macros

- #define __VEC_HEADER__

10.26.1 Detailed Description

Vector class declaration.

Author

Kailei Zhang, Chensong Zhang

Date

09/01/2019

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

10.26.2 Macro Definition Documentation

10.26.2.1 __VEC_HEADER__

```
#define __VEC_HEADER__
```

indicate VEC.hxx has been included before

10.27 VECUtil.cxx File Reference

Some auxiliary functions for VEC.

```
#include <cmath>
#include "VECUtil.hxx"
```

Functions

- FaspRetCode CheckVECSize (const VEC &v)

  Check whether the size of VEC object is zero.
- FaspRetCode CheckVECSize (const VEC &v1, const VEC &v2)

  Check whether two VEC sizes match.
- FaspRetCode CheckVECSize (const VEC &v, const INT &position)

  Check whether vector crossover.
- FaspRetCode CheckVECZero (const VEC &v, const DBL tol)

  Check whether there is a zero entry in VEC object.

10.27.1 Detailed Description

Some auxiliary functions for VEC.

Author

Kailei Zhang

Date

Sep/25/2019

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

10.28 VECUtil.hxx File Reference

Tools for checking and manipulating VEC.

```
#include "faspxx.hxx"
#include "VEC.hxx"
```

Macros

- #define __VECUTIL_HXX__

Functions

- FaspRetCode CheckVECSize (const VEC &v)

  Check whether the size of VEC object is zero.
- FaspRetCode CheckVECSize (const VEC &v1, const VEC &v2)

  Check whether two VEC sizes match.
- FaspRetCode CheckVECSize (const VEC &v, const INT &position)

  Check whether vector crossover.
- FaspRetCode CheckVECZero (const VEC &v, const DBL tol=SMALL_TOL)

  Check whether there is a zero entry in VEC object.

10.28.1 Detailed Description

Tools for checking and manipulating VEC.

Author

Chensong Zhang, Kailei Zhang

Date

Sep/24/2019

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

10.28.2 Macro Definition Documentation

10.28.2.1 __VECUTIL_HXX__

```
#define __VECUTIL_HXX__
```

indicate VECUtil.hxx has been included before

Index