FASP++

0.4.0 Feb/16/2020

Generated by Doxygen 1.8.17

Chapter 1

Introduction

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

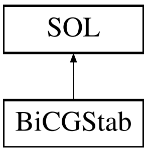
Class Documentation

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.

```
{
    // Nothing is needed for the moment!
}
```

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.

```
{
    const INT len = A.GetColSize();

    // Allocate memory for temporary vectors
    try {
        r0star.SetValues(len, 0.0);
        tmp.SetValues(len, 0.0);
        apj.SetValues(len, 0.0);
        asj.SetValues(len, 0.0);
        pj.SetValues(len, 0.0);
        rj.SetValues(len, 0.0);
        sj.SetValues(len, 0.0);
        ptmp.SetValues(len, 0.0);
        stmp.SetValues(len, 0.0);
        ms.SetValues(len, 0.0);
        mp.SetValues(len, 0.0);
        safe.SetValues(len, 0.0);
    } catch (std::bad_alloc &ex) {
        return FaspRetCode::ERROR_ALLOC_MEM;
    }

    // Set method type
    SetSolType(SOLType::BICGSTAB);

    // Setup the coefficient matrix
    this->A = &A;

    // Print used parameters
    if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);

    return FaspRetCode::SUCCESS;
}
```

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.

```
{
    if ( params.verbose > PRINT_NONE ) std::cout « "Use BiCGStab to solve Ax=b ...\n";

    // Check whether vector space sizes match
    if ( x.GetSize() != A->GetColSize() || b.GetSize() != A->GetRowSize()
                                        || A->GetRowSize() != A->GetColSize() )
        return FaspRetCode::ERROR_NONMATCH_SIZE;

    FaspRetCode errorCode = FaspRetCode::SUCCESS;

    // Declaration and definition of local variables
    const INT len = b.GetSize();
    const int maxStag = MAX_STAG_NUM; // maximum number of stagnation before quit
    const double solStagTol = 1e-4 * params.relTol; // solution stagnation tolerance

    int stagStep = 0, moreStep = 0;
    double resAbs = 1.0, resRel = 1.0, denAbs = 1.0, ratio = 0.0, resAbsOld = 1.0;
    double alpha, beta, rjr0star, rjr0startmp, omega, tmp12;

    PrintHead();

    // Initialize iterative method
    numIter = 0;
    A->Apply(x, this->tmp); // A * x -> tmp
    this->rj.WAXPBY(1.0, b, -1.0, this->tmp);

    // Prepare for the main loop
    this->r0star = this->rj;  // r0_{*} = r0c
    this->pj     = this->rj;  // p0 = r0

    // Main BiCGStab loop
    while ( numIter < params.maxIter ) {

        // Start from minIter instead of 0
        if ( numIter == params.minIter ) {
            resAbs = rj.Norm2();
            denAbs = (CLOSE_ZERO > resAbs) ? CLOSE_ZERO : resAbs;
            resRel = resAbs / denAbs;
            if ( resRel < params.relTol || resAbs < params.absTol ) break;
        }

        if ( numIter >= params.minIter ) PrintInfo(numIter, resRel, resAbs, ratio);

        //----------------------------------------------
        // BiCGStab iteration starts from here
        //----------------------------------------------

        ++numIter; // iteration count

        /* alpha_{j} = (rj,r0star)/(P * A * pj,r0star) */
        rjr0star = this->rj.Dot(this->r0star);

        /* main computational work */
        A->Apply(this->pj, this->apj);
        ptmp.SetValues(len,0.0);
        pc->Solve(this->apj, this->ptmp);

        tmp12 = this->ptmp.Dot(this->r0star);
        if ( fabs(tmp12) > 1e-40 ) alpha = rjr0star / tmp12;
        else {
            FASPXX_WARNING("Divided by zero!") // Possible breakdown
            errorCode = FaspRetCode::ERROR_DIVIDE_ZERO;
            break;
        }

        // sj = rj - alpha_{j} * P * A * p_{j}
        this->sj.WAXPBY(1.0, this->rj, -alpha, this->ptmp);

        // omega_j = (P * A * sj,sj)/(P * A * sj,P * A * sj)
        A->Apply(this->sj, this->asj);
        stmp.SetValues(len,0.0);
        pc->Solve(this->asj, this->stmp);
        omega = this->stmp.Dot(this->sj) / this->stmp.Dot(this->stmp);

        /* Update solution and residual */
        // x_{j+1} = x_{j} + alpha_{j} * P * pj + omega_j * P * s_{j}
        mp.SetValues(len,0.0);
        pc->Solve(this->pj, this->mp);
        ms.SetValues(len,0.0);
        pc->Solve(this->sj, this->ms);
        this->tmp.WAXPBY(alpha, this->mp, omega, this->ms);
        x.XPAY(1.0, this->tmp);

        // r_{j+1} = sj - omega_j * P * A * sj
        this->rj.WAXPBY(1.0, this->sj, -omega, this->stmp);

        //----------------------------------------------
        // One step of BiCGStab iteration ends here
        //----------------------------------------------

        // Apply several checks for safety
        if ( numIter >= params.minIter ) {
            // Compute norm of residual and output iteration information if needed
            resAbs = rj.Norm2();
            resRel = resAbs / denAbs;
            ratio  = resAbs / resAbsOld;

            // Save the best solution so far
            if ( numIter >= params.safeIter && resAbs < resAbsOld ) safe = x;

            // Apply stagnation checks if it converges slowly
            if ( ratio > KSM_CHK_RATIO && numIter > params.minIter ) {
                // Check I: if solution is close to zero, return ERROR_SOLVER_SOLSTAG
                double xNorminf = x.NormInf();
                if ( xNorminf < solStagTol ) {
                    if ( params.verbose > PRINT_MIN )
                        FASPXX_WARNING("Iteration stopped due to x vanishes!")
                    errorCode = FaspRetCode::ERROR_SOLVER_SOLSTAG;
                    break;
                }

                // Check II: if relative difference stagnated, try to restart
                double xRelDiff = fabs(alpha) * this->pj.Norm2() / x.Norm2();
                if ( (stagStep <= maxStag) && (xRelDiff < solStagTol) ) {
                    // Compute and update the residual before restart
                    A->Apply(x, this->rj);
                    this->rj.XPAY(-1.0, b);
                    resAbs = this->rj.Norm2();
                    resRel = resAbs / denAbs;
                    if ( params.verbose > PRINT_SOME ) {
                        FASPXX_WARNING("Possible iteration stagnate!")
                        WarnRealRes(resRel);
                    }

                    if ( resRel < params.relTol || resAbs < params.absTol ) break;
                    else {
                        if ( stagStep >= maxStag ) {
                            if ( params.verbose > PRINT_MIN )
                                FASPXX_WARNING("Iteration stopped due to stagnation!")
                            errorCode = FaspRetCode::ERROR_SOLVER_STAG;
                            break;
                        }
                        this->pj.SetValues(len, 0.0);
                        ++stagStep;
                    }

                    if ( params.verbose > PRINT_SOME ) {
                        WarnDiffRes(xRelDiff, resRel);
                        FASPXX_WARNING("Iteration restarted due to stagnation!")
                    }
                } // End of stagnation check!
            } // End of check I and II

            // Check III: prevent false convergence
            if ( resRel < params.relTol ) {
                // Compute true residual r = b - Ax and update residual
                A->Apply(x, this->rj);
                this->rj.XPAY(-1.0, b);

                // Compute residual norms and check convergence
                double resRelOld = resRel;
                resAbs = rj.Norm2();
                resRel = resAbs / denAbs;
                if ( resRel < params.relTol || resAbs < params.absTol ) break;

                if ( params.verbose >= PRINT_MORE ) {
                    FASPXX_WARNING("False convergence!")
                    WarnCompRes(resRelOld);
                    WarnRealRes(resRel);
                }

                if ( moreStep >= params.restart ) {
                    // Note: restart has different meaning here
                    if ( params.verbose > PRINT_MIN )
                        FASPXX_WARNING("The tolerance might be too small!")
                    errorCode = FaspRetCode::ERROR_SOLVER_TOLSMALL;
                    break;
                }

                // Prepare for restarting method
                this->pj.SetValues(len, 0.0);
                ++moreStep;
            } // End of check!
        }

        // Prepare for the next iteration
        if ( numIter < params.maxIter ) {
            // Save residual for next iteration
            resAbsOld = resAbs;

            // beta_j = (r_{j+1},r0^{*}) / (r_{j},r0^{*}) * alpha_j / omega_j
            rjr0startmp = rjr0star;
            rjr0star = this->rj.Dot(this->r0star);
            beta = rjr0star / rjr0startmp * alpha / omega;

            // p_{j+1} = r_{j+1} + beta_j * (p_{j} - omega_j * P * A * p_{j})
            this->tmp.WAXPBY(1.0, this->pj, -omega, this->ptmp);
            this->pj.WAXPBY(1.0, this->rj, beta, this->tmp);
        }

    } // End of main BiCGStab loop

    // If minIter == numIter == maxIter (preconditioner only), skip this
    if ( not (numIter == params.minIter && numIter == params.maxIter) ) {
        this->norm2 = resAbs;
        this->normInf = rj.NormInf();
        PrintFinal(numIter, resRel, resAbs, ratio);
    }

    // Restore the saved best iteration if needed
    if ( numIter > params.safeIter ) x = safe;

    return errorCode;
}
```

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

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

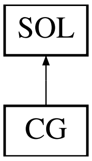
- BiCGStab.hxx
- BiCGStab.cxx

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.

```
{
    // Nothing is needed for the moment!
}
```

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.

```
{
    const INT len = A.GetColSize();
    SetSolType(SOLType::CG); // method type

    // Allocate memory for temporary vectors
    try {
        zk.SetValues(len, 0.0);
        pk.SetValues(len, 0.0);
        rk.SetValues(len, 0.0);
        ax.SetValues(len, 0.0);
        safe.SetValues(len, 0.0);
    } catch (std::bad_alloc &ex) {
        return FaspRetCode::ERROR_ALLOC_MEM;
    }

    // Setup the coefficient matrix
    this->A = &A;

    // Print used parameters
    if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);

    return FaspRetCode::SUCCESS;
}
```

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.

```
{
    if ( params.verbose > PRINT_NONE ) std::cout « "Use CG to solve Ax=b ...\n";

    // Check whether vector space sizes match
    if ( x.GetSize() != A->GetColSize() || b.GetSize() != A->GetRowSize()
        || A->GetRowSize() != A->GetColSize() )
        return FaspRetCode::ERROR_NONMATCH_SIZE;

    FaspRetCode errorCode = FaspRetCode::SUCCESS;

    // Local variables
    const INT len = b.GetSize();
    const int maxStag = MAX_STAG_NUM; // max number of stagnation checks
    const double solStagTol = 1e-4 * params.relTol; // solution stagnation tolerance
    const double solZeroTol = CLOSE_ZERO; // solution close to zero tolerance

    int stagStep = 0, moreStep = 0;
    double resAbs = 1.0, resRel = 1.0, denAbs = 1.0, ratio = 0.0, resAbsOld = 1.0;
    double alpha, beta, tmpa, tmpb;

    PrintHead();

    // Initialize iterative method
    numIter = 0;
    A->Apply(x, rk); // A * x -> rk
    rk.XPAY(-1.0, b); // b - rk -> rk

    // Preconditioned search direction
    zk.SetValues(len,0.0);
    pc->Solve(rk, zk); // preconditioning: B(r_k) -> z_k

    // Prepare for the main loop
    pk = zk;
    tmpa = zk.Dot(rk);

    // Main CG loop
    while ( numIter < params.maxIter ) {

        // Start from minIter instead of 0
        if ( numIter == params.minIter ) {
            resAbs = rk.Norm2();
            denAbs = (CLOSE_ZERO > resAbs) ? CLOSE_ZERO : resAbs;
            resRel = resAbs / denAbs;
            if (resRel < params.relTol || resAbs < params.absTol) break;
        }

        if ( numIter >= params.minIter ) PrintInfo(numIter, resRel, resAbs, ratio);

        //----------------------------------------------
        // CG iteration starts from here
        //----------------------------------------------

         ++numIter; // iteration count

         A->Apply(pk, ax); // ax = A * p_k, main computational work

         // alpha_k = (z_{k-1}, r_{k-1})/(A*p_{k-1},p_{k-1})
         tmpb = ax.Dot(pk);
         if ( fabs(tmpb) > CLOSE_ZERO * CLOSE_ZERO )
             alpha = tmpa / tmpb;
         else {
             FASPXX_WARNING("Divided by zero!")
             errorCode = FaspRetCode::ERROR_DIVIDE_ZERO;
             break;
         }

         // Update solution and residual
         x.AXPY(alpha, pk);   // x_k = x_{k-1} + alpha_k*p_{k-1}
         rk.AXPY(-alpha, ax); // r_k = r_{k-1} - alpha_k*A*p_{k-1}

         //----------------------------------------------
         // One step of CG iteration ends here
         //----------------------------------------------

         // Apply several checks for robustness
         if ( numIter >= params.minIter ) {
             // Compute norm of residual and output iteration information if needed
             resAbs = rk.Norm2();
             resRel = resAbs / denAbs;
             ratio  = resAbs / resAbsOld; // convergence ratio between two steps

             // Save the best solution so far
             if ( numIter >= params.safeIter && resAbs < resAbsOld ) safe = x;

             // Apply stagnation checks if it converges slowly
             if ( ratio > KSM_CHK_RATIO ) {
                 // Check I: if solution is close to zero, return ERROR_SOLVER_SOLSTAG
                 double xNormInf = x.NormInf();
                 if (xNormInf < solZeroTol) {
                     if (params.verbose > PRINT_MIN)
                         FASPXX_WARNING("Iteration stopped due to x vanishes!")
                     errorCode = FaspRetCode::ERROR_SOLVER_SOLSTAG;
                     break;
                 }

                 // Check II: if relative difference close to zero, try to restart
                 double xRelDiff = fabs(alpha) * this->pk.Norm2() / x.Norm2();
                 if ( (stagStep <= maxStag) && (xRelDiff < solStagTol) ) {
                     // Compute and update the residual before restart
                     A->Apply(x, this->rk);
                     this->rk.XPAY(-1.0, b);
                     resAbs = this->rk.Norm2();
                     resRel = resAbs / denAbs;
                     if ( params.verbose > PRINT_SOME ) {
                         FASPXX_WARNING("Possible iteration stagnate!")
                         WarnRealRes(resRel);
                     }

```

```
                    if ( resRel < params.relTol || resAbs < params.absTol ) break;
                    else {
                        if ( stagStep >= maxStag ) {
                            if ( params.verbose > PRINT_MIN )
                                FASPXX_WARNING("Iteration stopped due to stagnation!")
                            errorCode = FaspRetCode::ERROR_SOLVER_STAG;
                            break;
                        }
                        this->pk.SetValues(len, 0.0);
                        ++stagStep;
                    }

                    if ( params.verbose > PRINT_SOME ) {
                        WarnDiffRes(xRelDiff, resRel);
                        FASPXX_WARNING("Iteration restarted due to stagnation!")
                    }
                } // End of stagnation check!
            } // End of check I and II

            // Check III: prevent false convergence!!!
            if ( resRel < params.relTol ) {
                // Compute and update the true residual r = b - Ax
                A->Apply(x, this->rk);
                this->rk.XPAY(-1.0, b);

                // Compute residual norms and check convergence
                double resRelOld = resRel;
                resAbs = rk.Norm2();
                resRel = resAbs / denAbs;
                if ( resRel < params.relTol || resAbs < params.absTol ) break;

                // If false converged, print out warning messages
                if ( params.verbose >= PRINT_MORE ) {
                    FASPXX_WARNING("False convergence!")
                    WarnCompRes(resRelOld);
                    WarnRealRes(resRel);
                }

                if ( moreStep >= params.restart ) {
                    // Note: restart has different meaning here
                    if ( params.verbose > PRINT_MIN )
                        FASPXX_WARNING("The tolerance is too small!")
                    errorCode = FaspRetCode::ERROR_SOLVER_TOLSMALL;
                    break;
                }

                // Prepare for restarting method
                this->pk.SetValues(0.0);
                ++moreStep;
            } // End of check!
        }

        // Prepare for the next iteration
        if ( numIter < params.maxIter ) {
            // Save the residual for next iteration
            resAbsOld = resAbs;

            // Apply preconditioner z_k = B(r_k)
            zk.SetValues(len,0.0);
            pc->Solve(rk, zk);

            // Compute beta_k = (z_k, r_k) / (z_{k-1}, r_{k-1})
            tmpb = zk.Dot(rk);
            beta = tmpb / tmpa;
            tmpa = tmpb;

            // Compute p_k = z_k + beta_k*p_{k-1}
            pk.XPAY(beta, zk);
        }

    } // End of main CG loop

    // If minIter == numIter == maxIter (preconditioner only), skip this
    if ( not (numIter == params.minIter && numIter == params.maxIter) ) {
        this->norm2 = resAbs;
        this->normInf = rk.NormInf();
        PrintFinal(numIter, resRel, resAbs, ratio);
    }

    // Restore the saved best iteration if needed
    if ( numIter > params.safeIter ) x = safe;

    return errorCode;
}
```

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

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

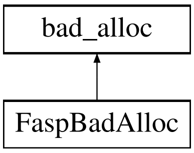
- CG.hxx
- CG.cxx

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:

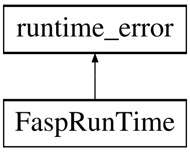
- RetCode.hxx
- RetCode.cxx

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:

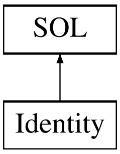
- Timing.hxx

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.

```
{
    x = b;
    return FaspRetCode::SUCCESS;
}
```

References SUCCESS.

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

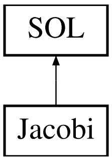
- Iter.hxx
- Iter.cxx

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.

```
{
    const INT len = A.GetColSize();
    SetSolType(SOLType::Jacobi); // method type

    // Allocate memory for temporary vectors
    try {
        rk.SetValues(len, 0.0);
    } catch (std::bad_alloc &ex) {
        return FaspRetCode::ERROR_ALLOC_MEM;
    }

    // Get diagonal and compute its reciprocal
    A.GetDiag(diagInv);
    diagInv.Reciprocal();

    // Setup the coefficient matrix
    this->A = &A;
    this->omega = params.weight;

    // Print used parameters if necessary
    if ( params.verbose > PRINT_MIN ) PrintParam(std::cout);

    return FaspRetCode::SUCCESS;
}
```

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:

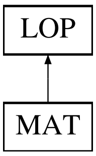
- Iter.hxx
- Iter.cxx

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.

```
{
    this->nrow = nrow;
    this->mcol = mcol;
}
```

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.

```
{
    this->nrow = nrow;
    this->mcol = nrow;
}
```

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.

```
{
    this->nrow = lop.nrow;
    this->mcol = lop.mcol;
}
```

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.

```
{
    return this->mcol;
}
```

References mcol.

9.9.3.2 GetRowSize()

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

Get row space dimension.

Dimension of the row space of LOP.

```
{
    return this->nrow;
}
```

References nrow.

9.9.3.3 operator=()

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

Overload the = operator.

Assignment for the LOP object.

```
{
    this->nrow = lop.nrow;
    this->mcol = lop.mcol;
    return *this;
}
```

References mcol, and nrow.

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

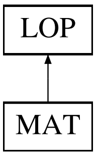
- LOP.hxx
- LOP.cxx

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.

```
                                                                         {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->values = values;
    this->colInd = colInd;
    this->rowPtr = rowPtr;
    this->diagPtr = diagPtr;
}
```

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.

```
                                          {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->values = values;
    this->colInd = colInd;
    this->rowPtr = rowPtr;
    this->FormDiagPtr();
}
```

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.

```
                                                                          {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->colInd = colInd;
    this->rowPtr = rowPtr;
    this->values.resize(0);
    this->FormDiagPtr();
}
```

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.

```
                                        {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->colInd = colInd;
    this->rowPtr = rowPtr;
    this->diagPtr = diagPtr;
    this->values.resize(0);
}
```

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.

```
                      {
    INT size = v.GetSize();

    // Return an empty matrix if size==0
    if (size == 0) {
        this->Empty();
        return;
    }

    // Set MAT size
    this->nrow = size;
    this->mcol = size;
    this->nnz = size;

    INT *p;
    try {
        p = new INT[size + 1];
    } catch (std::bad_alloc &ex) {
        this->nrow = 0;
        this->mcol = 0;
        this->nnz = 0;
        throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
    }

    // Set values from v
    this->values.resize(size);
    for (INT j = 0; j < size; ++j) this->values[j] = v.values[j];

    // Set colInd to {0, 1, ..., size-1}
    for (INT j = 0; j <= size; ++j) p[j] = j;
    this->colInd.resize(size);
    this->colInd.assign(p, p + size);

    // Set rowPtr to {0, 1, ..., size}
    this->rowPtr.resize(size + 1);
    this->rowPtr.assign(p, p + size + 1);

    // Set diagPtr to {0, 1, ..., size-1}
    this->diagPtr.resize(size);
    this->diagPtr.assign(p, p + size);

    delete[] p;
}
```

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.

```
                                 {
    const INT size = vt.size();

    // Return an empty matrix if size==0
    if (size == 0) {
        this->Empty();
        return;
    }

    // Set MAT size
    this->nrow = size;
    this->mcol = size;
    this->nnz = size;

    INT *p;
    try {
        p = new INT[size + 1];
    } catch (std::bad_alloc &ex) {
        this->nrow = 0;
        this->mcol = 0;
        this->nnz = 0;
        throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
    }

    // Set values from vt
    this->values.resize(size);
    this->values.assign(vt.begin(), vt.begin() + size);

    // Set colInd to {0, 1, ..., size-1}
    for (INT j = 0; j <= size; ++j) p[j] = j;
    this->colInd.resize(size);
    this->colInd.assign(p, p + size);

    // Set rowPtr to {0, 1, ..., size}
    this->rowPtr.resize(size + 1);
    this->rowPtr.assign(p, p + size + 1);

    // Set diagPtr to {0, 1, ..., size-1}
    this->diagPtr.resize(size);
    this->diagPtr.assign(p, p + size);

    delete[] p;
}
```

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.

```
                        {
    this->nrow = mat.nrow;
    this->mcol = mat.mcol;
    this->nnz = mat.nnz;
    this->values = mat.values;
    this->colInd = mat.colInd;
    this->rowPtr = mat.rowPtr;
    this->diagPtr = mat.diagPtr;
}
```

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.

```
                                                                       {

    MAT tmpMat;
    INT i, j, k, l;
    INT count = 0, added, countrow;

    if (mat1.nnz == 0) {
        tmpMat = mat2;
        tmpMat.Scale(b);
        return;
    }

    if (mat2.nnz == 0) {
        tmpMat = mat1;
        tmpMat.Scale(a);
        return;
    }

    tmpMat.nrow = mat1.nrow;
    tmpMat.mcol = mat1.mcol;

    tmpMat.rowPtr.resize(tmpMat.nrow + 1);
    tmpMat.colInd.resize(mat1.nnz + mat2.nnz);
    tmpMat.values.resize(mat1.nnz + mat2.nnz);

    tmpMat.colInd.assign(mat1.nnz + mat2.nnz, -1);

    for (i = 0; i < mat1.nrow; ++i) {
        countrow = 0;
        for (j = mat1.rowPtr[i]; j < mat1.rowPtr[i + 1]; ++j) {
            tmpMat.values[count] = a * mat1.values[j];
            tmpMat.colInd[count] = mat1.colInd[j];
            ++tmpMat.rowPtr[i + 1];
            ++count;
            ++countrow;
        }

        for (k = mat2.rowPtr[i]; k < mat2.rowPtr[i + 1]; ++k) {
            added = 0;
            for (l = tmpMat.rowPtr[i]; l < tmpMat.rowPtr[i] + countrow + 1; ++l) {
                if (mat2.colInd[k] == tmpMat.colInd[l]) {
                    tmpMat.values[l] = tmpMat.values[l] + b * mat2.values[k];
                    added = 1;
                    break;
                }
            }
            if (added == 0) {
                tmpMat.values[count] = b * mat2.values[k];
                tmpMat.colInd[count] = mat2.colInd[k];
                ++tmpMat.rowPtr[i + 1];
                ++count;
            }
        }
        tmpMat.rowPtr[i + 1] += tmpMat.rowPtr[i];
    }
    tmpMat.nnz = count;
    tmpMat.colInd.resize(count);
    tmpMat.values.resize(count);
    tmpMat.colInd.shrink_to_fit();
    tmpMat.values.shrink_to_fit();

    SortCSRRow(tmpMat.nrow, tmpMat.mcol, tmpMat.nnz, tmpMat.rowPtr, tmpMat.colInd,
               tmpMat.values);

    tmpMat.FormDiagPtr();
    *this = tmpMat;
}
```

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.

```
                                          {
    INT begin, i, k;

    if ( !this->values.empty() ) { // Regular sparse matrix
        for ( i = 0; i < this->nrow; ++i ) {
            begin = this->rowPtr[i];
            switch (this->rowPtr[i + 1] - begin) {
                case 4:
                    w.values[i] = this->values[begin]
                                  * v.values[this->colInd[begin]];
                    w.values[i] += this->values[begin + 1]
                                   * v.values[this->colInd[begin + 1]];
                    w.values[i] += this->values[begin + 2]
                                   * v.values[this->colInd[begin + 2]];
                    w.values[i] += this->values[begin + 3]
                                   * v.values[this->colInd[begin + 3]];
                    break;
                case 5:
                    w.values[i] = this->values[begin]
                                  * v.values[this->colInd[begin]];
                    w.values[i] += this->values[begin + 1]
                                   * v.values[this->colInd[begin + 1]];
                    w.values[i] += this->values[begin + 2]
                                   * v.values[this->colInd[begin + 2]];
                    w.values[i] += this->values[begin + 3]
                                   * v.values[this->colInd[begin + 3]];
                    w.values[i] += this->values[begin + 4]
                                   * v.values[this->colInd[begin + 4]];
                    break;
                case 6:
                    w.values[i] = this->values[begin]
                                  * v.values[this->colInd[begin]];
                    w.values[i] += this->values[begin + 1]
                                   * v.values[this->colInd[begin + 1]];
                    w.values[i] += this->values[begin + 2]
                                   * v.values[this->colInd[begin + 2]];
                    w.values[i] += this->values[begin + 3]
                                   * v.values[this->colInd[begin + 3]];
                    w.values[i] += this->values[begin + 4]
                                   * v.values[this->colInd[begin + 4]];
                    w.values[i] += this->values[begin + 5]
                                   * v.values[this->colInd[begin + 5]];
                    break;
                default:
                    w.values[i] =
                            this->values[begin] * v.values[this->colInd[begin]];
                    for (k = begin + 1; k < this->rowPtr[i + 1]; ++k)
                        w.values[i] += this->values[k] * v.values[this->colInd[k]];
            }
        }
    } else { // Only sparse structure
        for ( i = 0; i < this->nrow; ++i ) {
            begin = this->rowPtr[i];
            switch (this->rowPtr[i + 1] - begin) {
                case 4:
                    w.values[i] = v.values[this->colInd[begin]];
                    w.values[i] += v.values[this->colInd[begin + 1]];
                    w.values[i] += v.values[this->colInd[begin + 2]];
                    w.values[i] += v.values[this->colInd[begin + 3]];
                    break;
                case 5:
                    w.values[i] = v.values[this->colInd[begin]];
                    w.values[i] += v.values[this->colInd[begin + 1]];
                    w.values[i] += v.values[this->colInd[begin + 2]];
                    w.values[i] += v.values[this->colInd[begin + 3]];
                    w.values[i] += v.values[this->colInd[begin + 4]];
                    break;
                case 6:
                    w.values[i] = v.values[this->colInd[begin]];
                    w.values[i] += v.values[this->colInd[begin + 1]];
                    w.values[i] += v.values[this->colInd[begin + 2]];
                    w.values[i] += v.values[this->colInd[begin + 3]];
                    w.values[i] += v.values[this->colInd[begin + 4]];
                    w.values[i] += v.values[this->colInd[begin + 5]];
                    break;
                default:
                    w.values[i] = v.values[this->colInd[begin]];
                    for (k = begin + 1; k < this->rowPtr[i + 1]; ++k)
                        w.values[i] += v.values[this->colInd[k]];
            }
        }
    } // end if values.size > 0
}
```

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.

```
                               {
    mat = *this;
}
```

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.

```
                                 {
    m.nrow = this->nrow;
    m.mcol = this->mcol;
    m.nnz = this->nrow;

    m.rowPtr.resize(m.nrow + 1);
    for ( INT j = 0; j < m.nrow + 1; ++j ) m.rowPtr[j] = j;

    m.diagPtr.resize(m.nrow);
    for ( INT j = 0; j < m.nrow; ++j ) m.diagPtr[j] = j;

    m.colInd.resize(m.nnz);
    for ( INT j = 0; j < m.nnz; ++j ) m.colInd[j] = j;

    m.values.resize(m.nnz);
    for ( INT j = 0; j < m.nnz; ++j )
        m.values[j] = 1.0 / this->values[this->diagPtr[j]];
}
```

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.

```
                      {
    return this->nnz;
}
```

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.

```
                                                          {
    if ( this->colInd[this->rowPtr[irow]] <= jcol &&
         this->colInd[this->rowPtr[irow + 1] - 1] >= jcol ) {
        for ( INT j = this->rowPtr[irow]; j < this->rowPtr[irow + 1]; ++j ) {
            if ( jcol == this->colInd[j] ) {
                if ( this->values.empty() )
                    return 1.0; // sparse structure indicator
                else
                    return this->values[j];
            }
        }
    }
    return 0.0;
}
```

9.10.3.7 Mult()

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

∗this = matl ∗ matr

∗this = matl ∗ matr.

```
                                               {
    INT l, count;
    INT *tmp = new INT[matr.mcol];

    MAT mat;

    this->nrow = matl.nrow;
    this->mcol = matr.mcol;
    this->rowPtr.resize(this->nrow + 1);

    for (INT i = 0; i < matr.mcol; ++i) tmp[i] = -1;

    for (INT i = 0; i < this->nrow; ++i) {
        count = 0;
        for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
            for (INT j = matr.rowPtr[matl.colInd[k]];
                 j < matr.rowPtr[matl.colInd[k] + 1]; ++j) {
                for (l = 0; l < count; ++l) {
                    if (tmp[l] == matr.colInd[j]) break;
                }
                if (l == count) {
                    tmp[count] = matr.colInd[j];
                    ++count;
                }
            }
        }
        this->rowPtr[i + 1] = count;
        for (INT j = 0; j < count; ++j) tmp[j] = -1;
    }

    for (INT i = 0; i < this->nrow; ++i) this->rowPtr[i + 1] += this->rowPtr[i];

    INT count_tmp;

    this->colInd.resize(this->rowPtr[this->nrow]);

    for (INT i = 0; i < this->nrow; ++i) {
        count_tmp = 0;
        count = this->rowPtr[i];
        for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
            for (INT j = matr.rowPtr[matl.colInd[k]];
                 j < matr.rowPtr[matl.colInd[k] + 1]; ++j) {
                for (l = 0; l < count_tmp; ++l) {
                    if (tmp[l] == matr.colInd[j]) break;
                }
                if (l == count_tmp) {
                    this->colInd[count] = matr.colInd[j];
                    tmp[count_tmp] = matr.colInd[j];
                    ++count;
                    ++count_tmp;
                }
            }
        }

        for (INT j = 0; j < count_tmp; ++j) tmp[j] = -1;
    }

    delete[] tmp;

    this->values.resize(this->rowPtr[this->nrow]);

    for (INT i = 0; i < this->nrow; ++i) {
        for (INT j = this->rowPtr[i]; j < this->rowPtr[i + 1]; ++j) {
            this->values[j] = 0;
            for (INT k = matl.rowPtr[i]; k < matl.rowPtr[i + 1]; ++k) {
                for (l = matr.rowPtr[matl.colInd[k]];
                     l < matr.rowPtr[matl.colInd[k] + 1]; ++l) {
                    if (matr.colInd[l] == this->colInd[j])
                        this->values[j] += matl.values[k] * matr.values[l];
                }
            }
        }
    }

    this->nnz = this->rowPtr[this->nrow] - this->rowPtr[0];

    this->FormDiagPtr();
}
```

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

9.10.3.8 MultLeft()

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

∗this = ∗this ∗ mat

∗this = ∗this ∗ mat.

```
                                    {
    MAT tmp;
    tmp = *this;

    Mult(tmp, mat);
}
```

References Mult().

9.10.3.9 MultRight()

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

∗this = mat ∗ ∗this

∗this = mat ∗ ∗this.

```
                                     {
    MAT tmp;
    tmp = *this;

    Mult(mat, tmp);
}
```

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.

```
                                                                {
    const INT n = this->nrow, m = this->mcol, nnz = this->nnz;
    INT i, j, k, p;

    MAT tmp;
    tmp.nrow = m;
    tmp.mcol = n;
    tmp.nnz = nnz;

    try {
        tmp.rowPtr.resize(m + 1);
        tmp.colInd.resize(nnz);
    } catch (std::bad_alloc &ex) {
        throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
    }

    if ( !this->values.empty() ) {
        try {
            tmp.values.resize(nnz);
        } catch (std::bad_alloc &ex) {
            throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
        }
    } else {
        tmp.values.resize(0);
    }

    for ( j = 0; j < nnz; ++j ) {
        i = this->colInd[j];
        if ( i < m - 1 ) ++tmp.rowPtr[i + 2];
    }

    for ( i = 2; i <= m; ++i ) tmp.rowPtr[i] += tmp.rowPtr[i - 1];

    if ( !this->values.empty() ) {
        for (i = 0; i < n; ++i) {
            INT begin = this->rowPtr[i];
            for (p = begin; p < this->rowPtr[i + 1]; ++p) {
                j = this->colInd[p] + 1;
                k = tmp.rowPtr[j];
                tmp.colInd[k] = i;
                tmp.values[k] = this->values[p];
                tmp.rowPtr[j] = k + 1;
            }
        }
    } else {
        for (i = 0; i < n; ++i) {
            INT begin = this->rowPtr[i];
            for (p = begin; p < this->rowPtr[i + 1]; ++p) {
                j = this->colInd[p] + 1;
                k = tmp.rowPtr[j];
                tmp.colInd[k] = i;
                tmp.rowPtr[j] = k + 1;
            }
        }
    }

    v = v2;

    INT begin;
    for (i = 0; i < tmp.nrow; ++i) {
        begin = tmp.rowPtr[i];
        for (j = begin; j < this->rowPtr[i + 1]; ++j)
            v.values[i] += v1.values[tmp.colInd[j]] * tmp.values[j];
    }
}
```

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

9.10.3.11 operator=()

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

Overload = operator.

Assignment for the MAT object.

```
                                    {
    this->nrow = mat.nrow;
    this->mcol = mat.mcol;
    this->nnz = mat.nnz;
    this->values = mat.values;
    this->colInd = mat.colInd;
    this->rowPtr = mat.rowPtr;
    this->diagPtr = mat.diagPtr;
    return *this;
}
```

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.

```
                              {
    if ( this->values.empty() ) return; // MAT is a sparse structure!!!

    for ( INT j = 0; j < this->nnz; ++j ) this->values[j] *= a;
}
```

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.

```
                                                    {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->rowPtr = rowPtr;
    this->colInd = colInd;
    this->values = values;
    this->FormDiagPtr();
}
```

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.

```
                                                    {
    if (nrow == 0 || mcol == 0 || nnz == 0) {
        this->Empty();
        return;
    }

    this->nrow = nrow;
    this->mcol = mcol;
    this->nnz = nnz;
    this->values = values;
    this->rowPtr = rowPtr;
    this->colInd = colInd;
    this->diagPtr = diagPtr;
}
```

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.

```
                          {
    if ( this->values.empty() ) return; // MAT is a sparse structure!!!

    for ( INT j : this->diagPtr ) this->values[j] += a;
}
```

9.10.3.16 Transpose()

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

Transpose of the matrix.

Transpose *this in place.

```
                     {
    const INT n = this->nrow, m = this->mcol, nnz = this->nnz;
    INT i, j, k, p;

    MAT tmp;
    tmp.nrow = this->mcol;
    tmp.mcol = this->nrow;
    tmp.nnz = this->nnz;

    try {
        tmp.rowPtr.resize(this->mcol + 1);
        tmp.colInd.resize(nnz);
    } catch (std::bad_alloc &ex) {
        throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
    }

    if (!this->values.empty()) {
        try {
            tmp.values.resize(nnz);
        } catch (std::bad_alloc &ex) {
            throw( FaspBadAlloc(__FILE__, __FUNCTION__, __LINE__) );
        }
    } else {
        tmp.values.resize(0);
    }

    for ( INT j = 0; j < nnz; ++j ) {
        i = this->colInd[j];
        if ( i < m - 1 ) ++tmp.rowPtr[i + 2];
    }

    for ( i = 2; i <= m; ++i ) tmp.rowPtr[i] += tmp.rowPtr[i - 1];

    if ( !this->values.empty() ) {
        for ( i = 0; i < n; ++i ) {
            INT begin = this->rowPtr[i];
            for ( p = begin; p < this->rowPtr[i + 1]; ++p ) {
                j = this->colInd[p] + 1;
                k = tmp.rowPtr[j];
                tmp.colInd[k] = i;
                tmp.values[k] = this->values[p];
                tmp.rowPtr[j] = k + 1;
            }
        }
    } else {
        for ( i = 0; i < n; ++i ) {
            INT begin = this->rowPtr[i];
            for ( p = begin; p < this->rowPtr[i + 1]; ++p ) {
                j = this->colInd[p] + 1;
                k = tmp.rowPtr[j];
                tmp.colInd[k] = i;
                tmp.rowPtr[j] = k + 1;
            }
        }
    }

    tmp.FormDiagPtr();
    this->operator=(tmp);
}
```

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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

9.11.2.6 Parse()

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

Parse the parameters.

Main entrance point for reading and handling parameters.

```
{
    // Read parameters
    ReadFromCommandLine();
    ReadFromFile();

    // Save the original and then merge
    SaveUserParams(paramsUserOrg);
    MergeParams();
}
```

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.

```
{
    size_t max_len = 0;
    for ( const auto& itm: paramsUser ) {
        if ( itm.paramName.length() > max_len ) max_len = itm.paramName.length();
    }

    static std::string indent = "    ";
    out « "Parameters used in program:\n"
        « "---------------------------------------------\n";

    for ( auto& itm: paramsUser ) {
        out « indent « std::setw(max_len) « std::left « itm.paramName « " [";
        switch (itm.paramType) {
            case BoolType:
                out « std::boolalpha « *((bool *) (itm.paramPtr))
                    « std::resetiosflags(out.flags());
                break;
            case IntType:
                out « *((int*) itm.paramPtr);
                break;
            case DoubleType:
                out « *((double*) itm.paramPtr);
                break;
            case StringType:
                out « *((std::string*) itm.paramPtr);
                break;
            case OutputType:
                out « *((Output *) (itm.paramPtr));
                break;
        }
        out « "]\n";
    }
    out « ’\n’;
}
```

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.

```
{
    static const char *indent = "    ";
    static const char *types[] = {"<bool>", "<int>", "<double>", "<string>", "<Output>"};

    out « "Usage: " « argv[0] « " [options] ...\n"
        « "Options:\n" « indent « std::setw(21) « std::left
        « "-h, --help" « "              : print help information and exit\n";

    for ( const auto& prm: paramsUser ) {
        ParamType type = prm.paramType;
        out « indent « std::setw(12) « std::left « prm.paramName
            « " " « std::setw(8) « types[type];
        if ( prm.paramMarker == 0)      out « ", optional   ";
        else if ( prm.paramMarker == 1) out « ", required   ";
        else                            out « ", params file";
        if ( !prm.paramHelp.empty() ) out « " : " « prm.paramHelp;

        out « ", default = [";
        switch ( type ) {
            case BoolType:
                out « std::boolalpha « *(bool *) (prm.paramPtr)
                    « std::setiosflags(out.flags());
                break;
            case IntType:
                out « *(int *) (prm.paramPtr);
                break;
            case DoubleType:
                out « *(double *) (prm.paramPtr);
                break;
            case StringType:
                out « *(char **) (prm.paramPtr);
                break;
            case OutputType:
                out « *(Output *) (prm.paramPtr);
                break;
        }
        out « "]\n";
    }
}
```

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

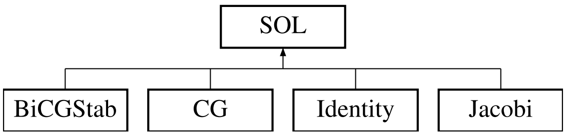
- Param.hxx
- Param.cxx

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.

```
{
    return this->normInf;
}
```

References normInf.

9.12.2.2 GetIterations()

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

Get number of iterations.

Get the value of numIter.

```
{
    return this->numIter;
}
```

References numIter.

9.12.2.3 GetNorm2()

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

Get Euclidean norm of residual.

Get L2 norm of the residual vector.

```
{
    return this->norm2;
}
```

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.

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

References params, and SOLParams::verbose.

9.12.2.5 SetAbsTol()

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

Set tolerance for absolute residual.

Set value for absTol.

```
{
    params.absTol = absTol;
}
```

References SOLParams::absTol, and params.

9.12.2.6 SetMaxIter()

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

Set max number of iterations.

Set value for maxIter.

```
{
    params.maxIter = maxIter;
}
```

References SOLParams::maxIter, and params.

9.12.2.7 SetMinIter()

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

Set min number of iterations.

Set value for minIter.

```
{
    params.minIter = minIter;
}
```

References SOLParams::minIter, and params.

9.12.2.8 SetOutput()

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

Set output level.

Set output level verbose.

```
{
    params.verbose = verbose;
}
```

References params, and SOLParams::verbose.

9.12.2.9 SetPC()

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

Setup preconditioner operator.

Build preconditioner operator.

```
{
    this->pc = &precond;
}
```

References pc.

9.12.2.10 SetRelTol()

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

Set tolerance for relative residual.

Set value for relTol.

```
{
    params.relTol = relTol;
}
```

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.

```
{
    params.restart = restart;
}
```

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.

```
{
    params.safeIter = safeIter;
}
```

References params, and SOLParams::safeIter.

9.12.2.13 SetSolType()

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

Set solver type.

Set SOLType.

```
{
    params.type = type;
}
```

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.

```
{
    for ( char & c : params.algName ) c = std::tolower(c); // Change to lowercase
    if ( params.algName == "cg" )
        params.type = SOLType::CG;
    else if ( params.algName == "bicgstab" )
        params.type = SOLType::BICGSTAB;
    else { // default solver type
        params.type = SOLType::CG;
        if ( params.verbose > PRINT_NONE )
            FASPXX_WARNING("Unknown solver type. Using CG instead!")
    }
}
```

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.

```
{
    params.weight = alpha;
}
```

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.

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

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.

```
{
    this->values = src;
    this->size = src.size();
}
```

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.

```
{
    this->values = src.values;
    this->size = src.size;
}
```

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.

```
{
    if ( src == nullptr || size == 0 ) {
        this->size = 0;
        return;
    }
    this->values.assign(src, src + size);
    this->size = size;
}
```

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.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     = fabs(this->values[i]);
        this->values[i + 1] = fabs(this->values[i + 1]);
        this->values[i + 2] = fabs(this->values[i + 2]);
        this->values[i + 3] = fabs(this->values[i + 3]);
    }
    for ( i = len; i < this->size; ++i ) this->values[i] = fabs(this->values[i]);
}
```

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.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    switch ( (a == 1.0) + 2 * (b == 1.0) ) {
        case 0:
            for ( i = 0; i < len; i += 4 ) {
                this->values[i]     = a * this->values[i]     + b * y.values[i];
                this->values[i + 1] = a * this->values[i + 1] + b * y.values[i + 1];
                this->values[i + 2] = a * this->values[i + 2] + b * y.values[i + 2];
                this->values[i + 3] = a * this->values[i + 3] + b * y.values[i + 3];
            }
            for ( i = len; i < this->size; ++i )
                this->values[i] = a * this->values[i] + b * y.values[i];
            break;

        case 1:
            for ( i = 0; i < len; i += 4 ) {
                this->values[i]     += b * y.values[i];
                this->values[i + 1] += b * y.values[i + 1];
                this->values[i + 2] += b * y.values[i + 2];
                this->values[i + 3] += b * y.values[i + 3];
            }
            for ( i = len; i < this->size; ++i ) this->values[i] += b * y.values[i];
            break;

        case 2:
            for ( i = 0; i < len; i += 4 ) {
                this->values[i]     = a * this->values[i]     + y.values[i];
                this->values[i + 1] = a * this->values[i + 1] + y.values[i + 1];
                this->values[i + 2] = a * this->values[i + 2] + y.values[i + 2];
                this->values[i + 3] = a * this->values[i + 3] + y.values[i + 3];
            }
            for ( i = len; i < this->size; ++i )
                this->values[i] = a * this->values[i] + y.values[i];
            break;

        case 3:
            for ( i = 0; i < len; i += 4 ) {
                this->values[i]     += y.values[i];
                this->values[i + 1] += y.values[i + 1];
                this->values[i + 2] += y.values[i + 2];
                this->values[i + 3] += y.values[i + 3];
            }
            for ( i = len; i < this->size; ++i ) this->values[i] += y.values[i];
    }
}
```

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.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     += a * x.values[i];
        this->values[i + 1] += a * x.values[i + 1];
        this->values[i + 2] += a * x.values[i + 2];
        this->values[i + 3] += a * x.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] += a * x.values[i];
}
```

9.14.3.4 CopyTo()

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

Copy ∗this to another VEC.

dst = ∗this.

```
{
    dst.values = this->values;
    dst.size = this->size;
}
```

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.

```
{
    INT i;
    DBL dot1 = 0.0, dot2 = 0.0, dot3 = 0.0, dot4 = 0.0;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        dot1 += this->values[i]     * v.values[i];
        dot2 += this->values[i + 1] * v.values[i + 1];
        dot3 += this->values[i + 2] * v.values[i + 2];
        dot4 += this->values[i + 3] * v.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) dot1 += this->values[i] * v.values[i];
    return (dot1 + dot2 + dot3 + dot4);
}
```

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.

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

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.

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

9.14.3.8 GetSize()

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

Get the size of ∗this.

Return the size of VEC.

```
{
    return this->size;
}
```

9.14.3.9 GetValue()

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

Get the value of (∗this)[position].

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

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

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!

```
{
    if ( size == 0 || this->size == 0 ) {
        array = nullptr;
        return;
    }
    for ( INT j = 0; j < size; ++j ) array[j] = this->values[index[j] % this->size];
}
```

9.14.3.11 Max()

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

Find maximal value.

Find maximal value, unroll long for loops.

```
{
    DBL max1 = SMALL, max2 = SMALL, max3 = SMALL, max4 = SMALL;

    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        if ( max1 < this->values[i] )     max1 = this->values[i];
        if ( max2 < this->values[i + 1] ) max2 = this->values[i + 1];
        if ( max3 < this->values[i + 2] ) max3 = this->values[i + 2];
        if ( max4 < this->values[i + 3] ) max4 = this->values[i + 3];
    }
    for ( i = len; i < this->size; ++i )
        if ( max1 < this->values[i] ) max1 = this->values[i];

    max1 = max1 >= max2 ? max1 : max2;
    max3 = max3 >= max4 ? max3 : max4;
    return max1 >= max3 ? max1 : max3;
}
```

References SMALL.

9.14.3.12 Min()

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

Find minimal value.

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

```
{
    DBL min1 = LARGE, min2 = LARGE, min3 = LARGE, min4 = LARGE;

    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        if (min1 > this->values[i])     min1 = this->values[i];
        if (min2 > this->values[i + 1]) min2 = this->values[i + 1];
        if (min3 > this->values[i + 2]) min3 = this->values[i + 2];
        if (min4 > this->values[i + 3]) min4 = this->values[i + 3];
    }
    for ( i = len; i < this->size; ++i )
        if (min1 > this->values[i]) min1 = this->values[i];

    min1 = min1 <= min2 ? min1 : min2;
    min3 = min3 <= min4 ? min3 : min4;
    return min1 <= min3 ? min1 : min3;
}
```

References LARGE.

9.14.3.13 Norm2()

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

Compute Euclidean norm.

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

```
{
    INT i;
    DBL tmp1 = 0.0, tmp2 = 0.0, tmp3 = 0.0, tmp4 = 0.0;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        tmp1 += std::pow(this->values[i], 2);
        tmp2 += std::pow(this->values[i + 1], 2);
        tmp3 += std::pow(this->values[i + 2], 2);
        tmp4 += std::pow(this->values[i + 3], 2);
    }
    for ( i = len; i < this->size; ++i ) tmp1 += std::pow(this->values[i], 2);

    return sqrt(tmp1 + tmp2 + tmp3 + tmp4);
}
```

9.14.3.14 NormInf()

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

Compute infinity norm.

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

```
{
    INT i;
    DBL tmpNorm1 = 0.0, tmpNorm2 = 0.0, tmpNorm3 = 0.0, tmpNorm4 = 0.0;
    DBL tmp1, tmp2, tmp3, tmp4;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        tmp1 = fabs(this->values[i]);
        if ( tmp1 > tmpNorm1 ) tmpNorm1 = tmp1;
        tmp2 = fabs(this->values[i + 1]);
        if ( tmp2 > tmpNorm2 ) tmpNorm2 = tmp2;
        tmp3 = fabs(this->values[i + 2]);
        if ( tmp3 > tmpNorm3 ) tmpNorm3 = tmp3;
        tmp4 = fabs(this->values[i + 3]);
        if ( tmp4 > tmpNorm4 ) tmpNorm4 = tmp4;
    }
    for ( i = len; i < this->size; ++i ) {
        tmp1 = fabs(this->values[i]);
        if ( tmp1 > tmpNorm1 ) tmpNorm1 = tmp1;
    }

    tmpNorm1 = tmpNorm1 >= tmpNorm2 ? tmpNorm1 : tmpNorm2;
    tmpNorm3 = tmpNorm3 >= tmpNorm4 ? tmpNorm3 : tmpNorm4;
    return (tmpNorm1 > tmpNorm3 ? tmpNorm1 : tmpNorm3);
}
```

9.14.3.15 operator+=()

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

Overload += operator.

Unroll for loops to speed up calculation.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     += v.values[i];
        this->values[i + 1] += v.values[i + 1];
        this->values[i + 2] += v.values[i + 2];
        this->values[i + 3] += v.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] += v.values[i];
    return *this;
}
```

9.14.3.16 operator-=()

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

Overload -= operator.

Unroll for loops to speed up calculation.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     -= v.values[i];
        this->values[i + 1] -= v.values[i + 1];
        this->values[i + 2] -= v.values[i + 2];
        this->values[i + 3] -= v.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] -= v.values[i];
    return *this;
}
```

9.14.3.17 operator=()

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

Overload the = operator.

Assignment for the VEC object.

```
{
    this->values = src.values;
    this->size = src.size;
    return *this;
}
```

9.14.3.18 operator[]() [1/2]

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

Overload the [] operator.

Regular [] operator, same behavior as array.

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

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.

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

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.

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     /= v.values[i];
        this->values[i + 1] /= v.values[i + 1];
        this->values[i + 2] /= v.values[i + 2];
        this->values[i + 3] /= v.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] /= v.values[i];
}
```

9.14.3.21 PointwiseMult()

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

Scale by a vector pointwise.

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

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     *= v.values[i];
        this->values[i + 1] *= v.values[i + 1];
        this->values[i + 2] *= v.values[i + 2];
        this->values[i + 3] *= v.values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] *= v.values[i];
}
```

9.14.3.22 Reciprocal()

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

Compute reciprocal pointwise.

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

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     = 1.0 / this->values[i];
        this->values[i + 1] = 1.0 / this->values[i + 1];
        this->values[i + 2] = 1.0 / this->values[i + 2];
        this->values[i + 3] = 1.0 / this->values[i + 3];
    }
    for ( i = len; i < this->size; ++i ) this->values[i] = 1 / this->values[i];
}
```

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.

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

9.14.3.24 Scale()

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

Scale by a scalar.

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

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     *= a;
        this->values[i + 1] *= a;
        this->values[i + 2] *= a;
        this->values[i + 3] *= a;
    }
    for ( i = len; i < this->size; ++i ) this->values[i] *= a;
}
```

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.

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

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.

```
{
    if ( array == nullptr || size == 0 ) {
        this->values.resize(0);
        this->size = 0;
        return;
    }
    this->values.assign(array, array + size);
    this->size = size;
}
```

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.

```
{
    this->values = src;
    this->size = src.size();
}
```

9.14.3.28 Shift()

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

Shift by a scalar pointwise.

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

```
{
    INT i;
    const INT len = this->size - this->size % 4;
    for ( i = 0; i < len; i += 4 ) {
        this->values[i]     += a;
        this->values[i + 1] += a;
        this->values[i + 2] += a;
        this->values[i + 3] += a;
    }
    for ( i = len; i < this->size; ++i ) this->values[i] += a;
}
```

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.

```
{
    INT i;
    this->size = v1.size;
    const INT len = this->size - this->size % 4;
    switch ( (a == 1) + 2 * (b == 1) ) {
        case 0:
            for ( i = 0; i < len; i += 4 ) {
                this->values[i]     = a * v1.values[i]     + b * v2.values[i];
                this->values[i + 1] = a * v1.values[i + 1] + b * v2.values[i + 1];
                this->values[i + 2] = a * v1.values[i + 2] + b * v2.values[i + 2];
                this->values[i + 3] = a * v1.values[i + 3] + b * v2.values[i + 3];
            }
            for ( i = len; i < this->size; ++i )
                this->values[i] = a * v1.values[i] + b * v2.values[i];
            break;

        case 1:
            for ( i = 0; i < len; i += 4 ) {
```

```
        this->values[i]     = v1.values[i]     + b * v2.values[i];
        this->values[i + 1] = v1.values[i + 1] + b * v2.values[i + 1];
        this->values[i + 2] = v1.values[i + 2] + b * v2.values[i + 2];
        this->values[i + 3] = v1.values[i + 3] + b * v2.values[i + 3];
      }
      for ( i = len; i < this->size; ++i )
        this->values[i] = v1.values[i] + b * v2.values[i];
      break;

    case 2:
      for ( i = 0; i < len; i += 4 ) {
        this->values[i]     = a * v1.values[i]     + v2.values[i];
        this->values[i + 1] = a * v1.values[i + 1] + v2.values[i + 1];
        this->values[i + 2] = a * v1.values[i + 2] + v2.values[i + 2];
        this->values[i + 3] = a * v1.values[i + 3] + v2.values[i + 3];
      }
      for ( i = len; i < this->size; ++i )
        this->values[i] = a * v1.values[i] + v2.values[i];
      break;

    case 3:
      for ( i = 0; i < len; i += 4 ) {
        this->values[i]     = v1.values[i]     + v2.values[i];
        this->values[i + 1] = v1.values[i + 1] + v2.values[i + 1];
        this->values[i + 2] = v1.values[i + 2] + v2.values[i + 2];
        this->values[i + 3] = v1.values[i + 3] + v2.values[i + 3];
      }
      for ( i = len; i < this->size; ++i )
        this->values[i] = v1.values[i] + v2.values[i];
      break;
  }
}
```

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.

```
{
  INT i;
  const INT len = this->size - this->size % 4;
  for ( i = 0; i < len; i += 4 ) {
    this->values[i]     = a * this->values[i]     + x.values[i];
    this->values[i + 1] = a * this->values[i + 1] + x.values[i + 1];
    this->values[i + 2] = a * this->values[i + 2] + x.values[i + 2];
    this->values[i + 3] = a * this->values[i + 3] + x.values[i + 3];
  }
  for ( i = len; i < this->size; ++i )
    this->values[i] = x.values[i] + a * this->values[i];
}
```

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.

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    SOL solver;
    solver.SetSolTypeFromName(params); // get solver type
    auto sol = &solver;

    switch (params.type) {
        case SOLType::CG :
            sol = new class CG();
            sol->SetOutput(params.verbose);
            sol->SetMaxIter(params.maxIter);
            sol->SetMinIter(params.minIter);
            sol->SetRestart(params.restart);
            sol->SetRelTol(params.relTol);
            sol->SetAbsTol(params.absTol);
            sol->SetSafeIter(params.safeIter);
            sol->Setup(A);
            sol->SetPC(pc);
            retCode = sol->Solve(b, x);
            break;
        case SOLType::BICGSTAB :
            sol = new class BiCGStab();
            sol->SetOutput(params.verbose);
            sol->SetMaxIter(params.maxIter);
            sol->SetMinIter(params.minIter);
            sol->SetRestart(params.restart);
            sol->SetRelTol(params.relTol);
            sol->SetAbsTol(params.absTol);
            sol->SetSafeIter(params.safeIter);
            sol->Setup(A);
            sol->SetPC(pc);
            retCode = sol->Solve(b, x);
            break;
        default: // should never reach here!!!
            if ( params.verbose > PRINT_NONE )
                FASPXX_WARNING("Unknown Krylov method type")
            std::cout « sol->GetSolType(params.type) « "is not supported!\n";
    }

    return retCode;
}
```

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.

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    SOL solver;
    solver.SetSolTypeFromName(params); // get solver type
    auto sol = &solver;

    switch (params.type) {
        case SOLType::CG :
            sol = new class CG();
            sol->SetOutput(params.verbose);
            sol->SetMaxIter(params.maxIter);
            sol->SetMinIter(params.minIter);
            sol->SetRestart(params.restart);
            sol->SetRelTol(params.relTol);
            sol->SetAbsTol(params.absTol);
            sol->SetSafeIter(params.safeIter);
            sol->Setup(A);
            sol->SetPC(pc);
            retCode = sol->Solve(b, x);
            break;
        case SOLType::BICGSTAB :
            sol = new class BiCGStab();
            sol->SetOutput(params.verbose);
            sol->SetMaxIter(params.maxIter);
            sol->SetMinIter(params.minIter);
            sol->SetRestart(params.restart);
            sol->SetRelTol(params.relTol);
            sol->SetAbsTol(params.absTol);
            sol->SetSafeIter(params.safeIter);
            sol->Setup(A);
            sol->SetPC(pc);
            retCode = sol->Solve(b, x);
            break;
        default: // should never reach here!!!
            if ( params.verbose > PRINT_NONE )
                FASPXX_WARNING("Unknown Krylov method type")
            std::cout « sol->GetSolType(params.type) « "is not supported!\n";
    }

    return retCode;
}
```

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.

```
                                           {
    std::ofstream out;
    out.open(filename);

    out « mat.nrow « " " « mat.mcol « " " « mat.nnz « "\n";
    for (INT j = 0; j < mat.nrow + 1; ++j) out « mat.rowPtr[j] « "\n";
    for (INT j = 0; j < mat.nnz; ++j) out « mat.colInd[j] « "\n";
    for (INT j = 0; j < mat.nnz; ++j) out « mat.values[j] « "\n";

    out.close();
}
```

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.

```
                                           {
    INT begin, end, j, k;
    std::ofstream out;
    out.open(filename);
    MAT tmp = mat;
    tmp.Transpose();

    out « tmp.nrow « " " « tmp.mcol « " " « tmp.nnz « "\n";
    for (j = 0; j < tmp.nrow; ++j) {
        begin = tmp.rowPtr[j];
        end = tmp.rowPtr[j + 1];
        for (k = begin; k < end; ++k)
            out « j « " " « tmp.colInd[j] « " " « tmp.values[j] « std::endl;
    }

    out.close();
}
```

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

```
{
    if ( row == 0 || col == 0 || nnz == 0 ) return FaspRetCode::SUCCESS;

    /*
     * some simple examinations about parameters
     * to judge whether they are CSRx' parameters
     */
    /*----------------  begin  ----------------*/
    INT count = 0;
    INT begin, end;

    if ( row != rowPtr.size() - 1 ) goto Return;

    if ( row <= 0 || col <= 0 ) goto Return;

    if (((row > col) ? col : row) != diagPtr.size()) goto Return;

    if ( nnz != colInd.size()) goto Return;

    if ( nnz != values.size()) goto Return;

    if ( nnz != rowPtr[rowPtr.size() - 1] ) goto Return;

    for ( INT j = 0; j < row; ++j ) {
        if ( rowPtr[j] >= rowPtr[j + 1] ) goto Return;
    }

    if ( rowPtr[0] < 0 || rowPtr[row] > nnz ) goto Return;

    for ( INT j = 0; j < row; ++j ) {
        begin = rowPtr[j];
        end = rowPtr[j + 1];
        if ( begin == end ) goto Return;

        if ( end == begin + 1 ) {
            if ( colInd[begin] != j ) goto Return;
        }

        if ( end > begin + 1 ) {
            for ( INT k = begin; k < end - 1; ++k ) {
                if ( colInd[k] >= colInd[k + 1] ) goto Return;
            }
            if ( 0 > colInd[begin] ) goto Return;

            if ( colInd[end - 1] >= col ) goto Return;
        }
    }

    for ( INT j = 0; j < row; ++j ) {
        begin = rowPtr[j];
        end = rowPtr[j + 1];
        for ( INT k = begin; k < end; ++k ) {
            if ( colInd[k] == j ) {
                if ( diagPtr[count] != k )
                    goto Return;
                else
                    ++count;
            }
        }
    }
    if ( count != diagPtr.size()) goto Return;

    return FaspRetCode::SUCCESS;

    Return: return FaspRetCode::ERROR_INPUT_PAR;
}
```

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.

```
{
    auto retCode = FaspRetCode::SUCCESS;

    std::vector<INT> rowPtrCSR;
    std::vector<INT> colIndCSR;
    std::vector<DBL> valuesCSR;

    // Convert data format from MTX to CSR
    MTXtoCSR(row, col, nnz, rowInd, colInd, values,valuesCSR, colIndCSR,rowPtrCSR);

    // Sort CSR matrix row by row
    SortCSRRow(row, col, nnz, rowPtrCSR, colIndCSR, valuesCSR);

    // Check whether diagonal is a nonzero position
    CSRtoMAT(row, col, nnz, valuesCSR, colIndCSR, rowPtrCSR, mat);

    return retCode;
}
```

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 > &row↩ Ind, 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

```
{
    if ( row == 0 || col == 0 || nnz == 0 ) return FaspRetCode::SUCCESS;

    /*
     * some simple examinations about parameters
     * to judge whether they are CSRx' parameters
     */
    /*----------------  begin  ----------------*/
    INT count = 0;
    INT begin, end;

    if ( row != rowPtr.size() - 1 ) goto Return;

    if ( row <= 0 || col <= 0 ) goto Return;

    if (((row > col) ? col : row) != diagPtr.size()) goto Return;

    if ( nnz != colInd.size()) goto Return;

    if ( nnz != values.size()) goto Return;

    if ( nnz != rowPtr[rowPtr.size() - 1] ) goto Return;

    for ( INT j = 0; j < row; ++j ) {
        if ( rowPtr[j] >= rowPtr[j + 1] ) goto Return;
    }

    if ( rowPtr[0] < 0 || rowPtr[row] > nnz ) goto Return;

    for ( INT j = 0; j < row; ++j ) {
        begin = rowPtr[j];
        end = rowPtr[j + 1];
        if ( begin == end ) goto Return;

        if ( end == begin + 1 ) {
            if ( colInd[begin] != j ) goto Return;
        }

        if ( end > begin + 1 ) {
            for ( INT k = begin; k < end - 1; ++k ) {
                if ( colInd[k] >= colInd[k + 1] ) goto Return;
            }
            if ( 0 > colInd[begin] ) goto Return;

            if ( colInd[end - 1] >= col ) goto Return;
        }
    }

    for ( INT j = 0; j < row; ++j ) {
        begin = rowPtr[j];
        end = rowPtr[j + 1];
        for ( INT k = begin; k < end; ++k ) {
            if ( colInd[k] == j ) {
                if ( diagPtr[count] != k )
                    goto Return;
                else
                    ++count;
            }
        }
    }
    if ( count != diagPtr.size()) goto Return;

    return FaspRetCode::SUCCESS;

    Return: return FaspRetCode::ERROR_INPUT_PAR;
}
```

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.

```
{
    auto retCode = FaspRetCode::SUCCESS;

    std::vector<INT> rowPtrCSR;
    std::vector<INT> colIndCSR;
    std::vector<DBL> valuesCSR;

    // Convert data format from MTX to CSR
    MTXtoCSR(row, col, nnz, rowInd, colInd, values,valuesCSR, colIndCSR,rowPtrCSR);

    // Sort CSR matrix row by row
    SortCSRRow(row, col, nnz, rowPtrCSR, colIndCSR, valuesCSR);

    // Check whether diagonal is a nonzero position
    CSRtoMAT(row, col, nnz, valuesCSR, colIndCSR, rowPtrCSR, mat);

    return retCode;
}
```

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

Copyright (C) 2019–present by the FASP++ team. All rights reserved.

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)

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    // Open the file to read
    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if (!in.is_open()) { // judge whether file is opened successfully
        std::cout « "Reading from disk file " « fileName « std::endl;
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Read the file in to a buffer
    in.seekg(0, std::ios::end);
    long long int length = in.tellg(); // compute total bytes ’s number
    in.seekg(0, std::ios::beg);

    char decimal[128];
    char *buffer, *next;

    // Allocate memory space for storing the whole file
    try { // catch the bad allocation if it happens
        buffer = new char[length];
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the whole file in bytes
    in.close(); // close the file stream

    // Read number of rows
    INT count = 0;
    long long int position = 0; // mark the position of file pointer
    while (true) {
        if (buffer[position] != ’\n’) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            decimal[count] = ’\0’; // mark the end of ’decimal’ string
            ++position;
            break;
        }
    }

    row = std::strtol(decimal, &next, 10);
    if ( row <= 0 ) { // prevent memory leaks if error happens
        retCode = FaspRetCode::ERROR_INPUT_PAR;
        delete[] buffer;
        return retCode;
    }
    col = row;

    // Read row pointers
    try { // catch bad allocation if it happens
        rowPtr.resize(row + 1);
    } catch (std::bad_alloc &ex) {
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Read the rowPtr of CSRx matrix
    long int locate = 0;
    count = 0;
    while ( true ) {
        if (buffer[position] != ’\n’) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            decimal[count] = ’\0’;
            count = 0;
            rowPtr[locate] = std::strtol(decimal, &next, 10);
            ++locate;
            if (locate == row + 1) break;
        }
    }

    // Allocate memory for colInd and values
    try { // catch bad allocation if it happens
        nnz = rowPtr[row] - rowPtr[0];
        colInd.resize(nnz);
        values.resize(nnz);
    } catch (std::bad_alloc &ex) {
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Read column indices
    locate = 0;
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            decimal[count] = '\0';
            count = 0;
            colInd[locate] = std::strtol(decimal, &next, 10);
            ++locate;
            if (locate == nnz) break;
        }
    }

    // Read values
    locate = 0;
    while ( true ) {
        if (buffer[position] != '\n' && buffer[position] != '\0') {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            if (buffer[position] == '\0') break;
            ++position;
            decimal[count] = '\0';
            count = 0;
            values[locate] = std::strtod(decimal, &next);
            ++locate;
        }
    }
    if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    // If the indices start from 1, we shift them to start from 0
    if ( rowPtr[0] == 1 ) {
        for (count = 0; count <= row; ++count) rowPtr[count]--;
        for (count = 0; count < nnz; ++count) colInd[count]--;
    }

    delete[] buffer; // clean up memory space

    return retCode;
}
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ERROR_OPEN_FILE, and SUCCESS.

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.

```
{
    const int len = strlen(fileName);
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    if ( len <= 4 ) {
        retCode = FaspRetCode::ERROR_INPUT_FILE;
        return retCode;
    }

    // Check the file extension
    char fileExt[4];
    for ( int i = 0; i < 3; ++i ) fileExt[i] = tolower(fileName[len - 3 + i]);
    fileExt[3] = '\0';

    int flag = 0; // Undefined file format
    if ( strcmp(fileExt, "csr" ) == 0)
        flag = 1; // CSR file
    else if ( strcmp(fileExt, "mtx" ) == 0)
        flag = 2; // MTX file

    INT row, col, nnz;
    std::vector<INT> rowPtr, colInd, rowInd;
    std::vector<DBL> values;

    switch ( flag ) {
        case 1:
            try {
                retCode = ReadCSR(fileName, row, col, nnz,
                                  rowPtr, colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Sort each row in ascending order
            try {
                retCode = SortCSRRow(row, col, nnz, rowPtr, colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Convert a MTX matrix to MAT
            try {
                retCode = CSRtoMAT(row, col, nnz, values, colInd, rowPtr, dst);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }
            break;

        case 2:
            try {
                retCode = ReadMTX(fileName, row, col, nnz, rowInd,
                                  colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Sort each row in ascending order
            try {
                retCode = MTXtoMAT(row, col, nnz, rowInd, colInd, values, dst);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }
            break;

        default:
            FASPXX_WARNING("Unknown file format detected!")
            retCode = FaspRetCode::ERROR_INPUT_FILE;
            break;
    }

    return retCode;
}
```

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)

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    // Open the file to read
    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if (!in.is_open()) { // check whether file is opened successfully
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Read the file in to a buffer
    in.seekg(0, std::ios::end);
    const long long int length = in.tellg();
    in.seekg(0, std::ios::beg);

    char decimal[128];
    char *buffer, *next;
    long long int position = 0; // position of file pointer

    // Allocate temp space for storing the whole file
    try { // catch bad allocation if it happens
        buffer = new char[length];
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the whole file in bytes
    in.close(); // close the file stream

    int count = 0; // number of bytes in the decimal
    int mark = 0; // which number of integer is reading
    while ( true ) { // read matrix ’s row, column, nnz
        if ( buffer[position] != ’ ’ && buffer[position] != ’\n’ ) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            decimal[count] = ’\0’;
            count = 0;
            ++mark;
            ++position;
            switch (mark) {
                case 1: // first, integer, number of rows
                    row = std::strtol(decimal, &next, 10); break;
                case 2: // second, integer, number of columns
                    col = std::strtol(decimal, &next, 10); break;
                case 3: // third, integer, number of nonzeros
                    nnz = std::strtol(decimal, &next, 10); break;
                default:
                    FASPXX_WARNING("Unknown input value!")
            }
        }
        if ( mark == 3 ) break; // skip the rest
    }

    // Allocate memory space to store row indices, column indices and values
    try { // catch the bad allocation if it happens
        rowInd.resize(nnz);
        colInd.resize(nnz);
        values.resize(nnz);
    } catch (std::bad_alloc &ex) {
        delete[] buffer; // if bad allocation happens, free up the memory space
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Put MTX data into rowInd, colInd, and values
    long int locate = 0; // mark the position in rowInd, colInd and values
    long int tmp = 0;
    while ( true ) {
        if (buffer[position] != ’ ’ && buffer[position] != ’\n’ &&
            buffer[position] != ’\0’) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            if (buffer[position] == ’ ’) continue; // multiple consecutive spaces
            decimal[count] = ’\0’; // mark the end of ’decimal’ string
            count = 0;
            ++tmp;
            locate = tmp / 3;
            switch (tmp % 3) {
                case 1: // first: integer, row index
                    rowInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
                case 2: // second: integer, column index
                    colInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
                case 0: // third: double, value
                    values[locate-1] = std::strtod(decimal, &next); break;
            }
            if (buffer[position] == '\0') break;
        }
    }

    if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    delete[] buffer; // clean up memory space

    return retCode;
}
```

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.

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if ( !in.is_open() ) { // check whether file is opened successfully
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Compute total number of bytes of file
    in.seekg(0, std::ios::end);
    const long long int length = in.tellg();
    in.seekg(0, std::ios::beg);

    char decimal[128]; // temporary storage for data
    long long int position = 0; // mark the position of file pointer
    long int count = 0, len;

    char *buffer, *next;
    try { // catch bad allocation error if it happens
        buffer = new char[length]; // allocate memory for buffer
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the total bytes of file
    in.close(); // close the file pointer

    // Read in the size of VEC object
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++position;
            ++count;
        } else {
            decimal[count] = '\0';
            count = 0;
            ++position;
            len = std::strtol(decimal, &next, 10);
            break;
        }
    }

    // Allocate memory space and initialize
    dst.SetValues(len, 0.0);

    // Read in the VEC object's entries
    long int locate = 0; // mark the element position
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++position;
            ++count;
        } else {
            decimal[count] = '\0';
            count = 0;
            ++position;
            dst[locate] = std::strtod(decimal, &next);
            ++locate;
        }
        if (buffer[position] == '\0') break;
    }

    if ( locate != len ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    delete[] buffer; // clean up memory space

    return retCode;
}
```

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

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)

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    // Open the file to read
    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if (!in.is_open()) { // judge whether file is opened successfully
        std::cout « "Reading from disk file " « fileName « std::endl;
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Read the file in to a buffer
    in.seekg(0, std::ios::end);
    long long int length = in.tellg(); // compute total bytes ’s number
    in.seekg(0, std::ios::beg);

    char decimal[128];
    char *buffer, *next;

    // Allocate memory space for storing the whole file
    try { // catch the bad allocation if it happens
        buffer = new char[length];
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the whole file in bytes
    in.close(); // close the file stream

    // Read number of rows
    INT count = 0;
    long long int position = 0; // mark the position of file pointer
    while (true) {
        if (buffer[position] != ’\n’) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            decimal[count] = ’\0’; // mark the end of ’decimal’ string
            ++position;
            break;
        }
    }

    row = std::strtol(decimal, &next, 10);
    if ( row <= 0 ) { // prevent memory leaks if error happens
        retCode = FaspRetCode::ERROR_INPUT_PAR;
        delete[] buffer;
        return retCode;
    }
    col = row;

    // Read row pointers
    try { // catch bad allocation if it happens
        rowPtr.resize(row + 1);
    } catch (std::bad_alloc &ex) {
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Read the rowPtr of CSRx matrix
    long int locate = 0;
    count = 0;
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            decimal[count] = '\0';
            count = 0;
            rowPtr[locate] = std::strtol(decimal, &next, 10);
            ++locate;
            if (locate == row + 1) break;
        }
    }

    // Allocate memory for colInd and values
    try { // catch bad allocation if it happens
        nnz = rowPtr[row] - rowPtr[0];
        colInd.resize(nnz);
        values.resize(nnz);
    } catch (std::bad_alloc &ex) {
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Read column indices
    locate = 0;
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            decimal[count] = '\0';
            count = 0;
            colInd[locate] = std::strtol(decimal, &next, 10);
            ++locate;
            if (locate == nnz) break;
        }
    }

    // Read values
    locate = 0;
    while ( true ) {
        if (buffer[position] != '\n' && buffer[position] != '\0') {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            if (buffer[position] == '\0') break;
            ++position;
            decimal[count] = '\0';
            count = 0;
            values[locate] = std::strtod(decimal, &next);
            ++locate;
        }
    }
    if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    // If the indices start from 1, we shift them to start from 0
    if ( rowPtr[0] == 1 ) {
        for (count = 0; count <= row; ++count) rowPtr[count]--;
        for (count = 0; count < nnz; ++count) colInd[count]--;
    }

    delete[] buffer; // clean up memory space

    return retCode;
}
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ERROR_OPEN_FILE, and SUCCESS.

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.

```
{
    const int len = strlen(fileName);
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    if ( len <= 4 ) {
        retCode = FaspRetCode::ERROR_INPUT_FILE;
        return retCode;
    }

    // Check the file extension
    char fileExt[4];
    for ( int i = 0; i < 3; ++i ) fileExt[i] = tolower(fileName[len - 3 + i]);
    fileExt[3] = '\0';

    int flag = 0; // Undefined file format
    if ( strcmp(fileExt, "csr" ) == 0)
        flag = 1; // CSR file
    else if ( strcmp(fileExt, "mtx" ) == 0)
        flag = 2; // MTX file

    INT row, col, nnz;
    std::vector<INT> rowPtr, colInd, rowInd;
    std::vector<DBL> values;

    switch ( flag ) {
        case 1:
            try {
                retCode = ReadCSR(fileName, row, col, nnz,
                                  rowPtr, colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Sort each row in ascending order
            try {
                retCode = SortCSRRow(row, col, nnz, rowPtr, colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Convert a MTX matrix to MAT
            try {
                retCode = CSRtoMAT(row, col, nnz, values, colInd, rowPtr, dst);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }
            break;

        case 2:
            try {
                retCode = ReadMTX(fileName, row, col, nnz, rowInd,
                                  colInd, values);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }

            // Sort each row in ascending order
            try {
                retCode = MTXtoMAT(row, col, nnz, rowInd, colInd, values, dst);
                if ( retCode < 0 )
                    throw( FaspRunTime(retCode, __FILE__, __FUNCTION__, __LINE__) );
            }
            catch (FaspRunTime &ex) {
                ex.LogExcep();
                break;
            }
            break;

        default:
            FASPXX_WARNING("Unknown file format detected!")
            retCode = FaspRetCode::ERROR_INPUT_FILE;
            break;
    }

    return retCode;
}
```

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)

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    // Open the file to read
    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if (!in.is_open()) { // check whether file is opened successfully
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Read the file in to a buffer
    in.seekg(0, std::ios::end);
    const long long int length = in.tellg();
    in.seekg(0, std::ios::beg);

    char decimal[128];
    char *buffer, *next;
    long long int position = 0; // position of file pointer

    // Allocate temp space for storing the whole file
    try { // catch bad allocation if it happens
        buffer = new char[length];
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the whole file in bytes
    in.close(); // close the file stream

    int count = 0; // number of bytes in the decimal
    int mark = 0; // which number of integer is reading
    while ( true ) { // read matrix ’s row, column, nnz
        if ( buffer[position] != ’ ’ && buffer[position] != ’\n’ ) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            decimal[count] = ’\0’;
            count = 0;
            ++mark;
            ++position;
            switch (mark) {
                case 1: // first, integer, number of rows
                    row = std::strtol(decimal, &next, 10); break;
                case 2: // second, integer, number of columns
                    col = std::strtol(decimal, &next, 10); break;
                case 3: // third, integer, number of nonzeros
```

```
                    nnz = std::strtol(decimal, &next, 10); break;
                default:
                    FASPXX_WARNING("Unknown input value!")
            }
        }
        if ( mark == 3 ) break; // skip the rest
    }

    // Allocate memory space to store row indices, column indices and values
    try { // catch the bad allocation if it happens
        rowInd.resize(nnz);
        colInd.resize(nnz);
        values.resize(nnz);
    } catch (std::bad_alloc &ex) {
        delete[] buffer; // if bad allocation happens, free up the memory space
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }

    // Put MTX data into rowInd, colInd, and values
    long int locate = 0; // mark the position in rowInd, colInd and values
    long int tmp = 0;
    while ( true ) {
        if (buffer[position] != ’ ’ && buffer[position] != ’\n’ &&
            buffer[position] != ’\0’) {
            decimal[count] = buffer[position];
            ++count;
            ++position;
        } else {
            ++position;
            if (buffer[position] == ’ ’) continue; // multiple consecutive spaces
            decimal[count] = ’\0’; // mark the end of ’decimal’ string
            count = 0;
            ++tmp;
            locate = tmp / 3;
            switch (tmp % 3) {
                case 1: // first: integer, row index
                    rowInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
                case 2: // second: integer, column index
                    colInd[locate] = std::strtol(decimal, &next, 10) - 1; break;
                case 0: // third: double, value
                    values[locate-1] = std::strtod(decimal, &next); break;
            }
            if (buffer[position] == ’\0’) break;
        }
    }

    if ( locate != nnz ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    delete[] buffer; // clean up memory space

    return retCode;
}
```

References ERROR_ALLOC_MEM, ERROR_INPUT_FILE, ERROR_OPEN_FILE, FASPXX_WARNING, and SUCCESS.

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.

```
{
    FaspRetCode retCode = FaspRetCode::SUCCESS;

    std::cout « "Reading from disk file " « fileName « std::endl;
    std::ifstream in(fileName);
    if ( !in.is_open() ) { // check whether file is opened successfully
        retCode = FaspRetCode::ERROR_OPEN_FILE;
        return retCode;
    }

    // Compute total number of bytes of file
    in.seekg(0, std::ios::end);
    const long long int length = in.tellg();
    in.seekg(0, std::ios::beg);

    char decimal[128]; // temporary storage for data
    long long int position = 0; // mark the position of file pointer
    long int count = 0, len;

    char *buffer, *next;
    try { // catch bad allocation error if it happens
        buffer = new char[length]; // allocate memory for buffer
    } catch (std::bad_alloc &ex) {
        in.close();
        retCode = FaspRetCode::ERROR_ALLOC_MEM;
        return retCode;
    }
    in.read(buffer, length); // read the total bytes of file
    in.close(); // close the file pointer

    // Read in the size of VEC object
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++position;
            ++count;
        } else {
            decimal[count] = '\0';
            count = 0;
            ++position;
            len = std::strtol(decimal, &next, 10);
            break;
        }
    }

    // Allocate memory space and initialize
    dst.SetValues(len, 0.0);

    // Read in the VEC object's entries
    long int locate = 0; // mark the element position
    while ( true ) {
        if (buffer[position] != '\n') {
            decimal[count] = buffer[position];
            ++position;
            ++count;
        } else {
            decimal[count] = '\0';
            count = 0;
            ++position;
            dst[locate] = std::strtod(decimal, &next);
            ++locate;
        }
        if (buffer[position] == '\0') break;
    }

    if ( locate != len ) retCode = FaspRetCode::ERROR_INPUT_FILE;

    delete[] buffer; // clean up memory space

    return retCode;
}
```

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.

```
{
    SUCCESS                 = 0,
    //---------- Input problems --------------------------------------------------//
    ERROR_OPEN_FILE         = -10,
    ERROR_INPUT_FILE        = -11,
    ERROR_INPUT_PAR         = -12,
    //---------- VEC or MAT data problems ----------------------------------------//
    ERROR_VEC_SIZE          = -14,
    ERROR_MAT_SIZE          = -15,
    ERROR_NONMATCH_SIZE     = -16,
    ERROR_MAT_DATA          = -17,
    ERROR_DIVIDE_ZERO       = -18,
    ERROR_MAT_ZERODIAG      = -19,
    //---------- Memory or function call problems --------------------------------//
    ERROR_ALLOC_MEM         = -20,
    ERROR_DUMMY_VAR         = -23,
    //---------- Iterative method problems ---------------------------------------//
    ERROR_SOLVER_TYPE       = -30,
    ERROR_SOLVER_PRECTYPE   = -31,
    ERROR_SOLVER_STAG       = -32,
    ERROR_SOLVER_SOLSTAG   = -33,
    ERROR_SOLVER_TOLSMALL  = -34,
    ERROR_SOLVER_MAXIT     = -39,
    //---------- AMG method problems ---------------------------------------------//
    ERROR_AMG_INTERP_TYPE  = -40,
    ERROR_AMG_SMOOTH_TYPE  = -41,
    ERROR_AMG_COARSE_TYPE  = -42,
    ERROR_AMG_COARSEING    = -43,
    ERROR_AMG_SETUP        = -49,
    //---------- ILU method problems ---------------------------------------------//
    ERROR_ILU_TYPE         = -50,
    ERROR_ILU_SETUP        = -59,
    //---------- ILU method problems ---------------------------------------------//
    ERROR_SWZ_TYPE         = -60,
    ERROR_SWZ_SETUP        = -69,
    //---------- Unknown problems (default) --------------------------------------//
    ERROR_UNKNOWN          = -99,
};
```

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.

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

References ERROR_ALLOC_MEM, ERROR_AMG_COARSE_TYPE, ERROR_AMG_COARSEING, ERROR_AMG_INTERP_TYPE, ERROR_AMG_SETUP, ERROR_AMG_SMOOTH_TYPE, ERROR_DIVIDE_ZERO, ERROR_DUMMY_VAR, ERROR_ILU_SETUP, ERROR_ILU_TYPE, ERROR_INPUT_FILE, ERROR_INPUT_PAR, ERROR_MAT_DATA, ERROR_MAT_SIZE, ERROR_MAT_ZERODIAG, ERROR_NONMATCH_SIZE, ERROR_OPEN_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.

```
             {
    CG        = 1,
    BICGSTAB  = 2,
    MINRES    = 3,
    GMRES     = 4,
    FGMRES    = 5,
    VFGMRES   = 6,
    Jacobi     = 11,
};
```

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