FASP User Guide

FASP Developer Team

Version 1.8.1

Contents

Chapter 1

Introduction

1.1 General description

The Fast Auxiliary Space Preconditioning (FASP) package provides source files (in ANSI C???) to build a library of iterative solvers and preconditioners for the solution of large scale linear systems of equations. The components of the base FASP library include several ready-to-use, modern, and efficient iterative solvers used in applications ranging from simple examples of discretized scalar partial differential equations (PDEs) to numerical simulations of complex, multicomponent physical systems via the Auxiliary Space Preconditioning framework [19].

The main components of the FASP basic library are:

- Basic linear iterative methods;
- Standard Krylov subspace methods;
- Geometric and Algebraic Multigrid (G/AMG) methods;
- Incomplete factorization methods.

The FASP distribution also includes several examples for solving simple benchmark problems.

1.2 Roadmap: from basics to complex applications

A distinct feature of the FASP software project is that it is an open ended project. It contains a basic kernel of sources and is maintained by a team of developers with the expertise to build efficient solvers for a wide range of complex numerical models.

As typical for an open-source software, the further development of FASP project will be largely based on the involvement of the community. We count on the users' input in providing requests for, as well as, contributions to, the expansion of FASP in different application areas. Our developers are ready to help in designing efficient solvers based on the FASP kernel for a large variety of numerical models.

The FASP software has been successfully used to build efficient solvers for several discretized PDEs and systems of PDEs: general scalar elliptic equations; linear elasticity; Brinkman equation; bi-harmonic equation; Stokes and Navier-Stokes equations; $H(\text{curl})/H(\text{div})$ systems; Maxwell's system. The resulting solvers have been applied in simulations from fluid dynamics, underground water simulation, fluid-structure interactions; Oldryod-B and Johnson-Seglman models; black-oil model in reservoir simulation; magnetohydrodynamics (MHD).

Several of these benchmark problems are included as examples in the this open-source distribution, others are under development or have more restrictive licensing.

1.3   How to use this guide

This user's guide describes how to use the existing solvers in FASP via a couple of simple tutorial problems. The user's guide is a self-contained document but does *not* provide any details about the algorithms or their implementation. Along with this guide, we provide a reference manual[1] for technical details on the implementation which includes references. We recommend that the users read these references to better understanding of the code. Furthermore, since FASP is under heavy development, please use this guide with caution because the code might have been changed before this document is updated.

1.4   How to obtain FASP

Chensong, can you change this below to just reflect the source-forge sources?

All the FASP packages are hosted on *BitBucket.org*[2] using Mercurial (Hg)[3]. A Hg client for GNU Linux, Mac OS X, or Windows can be downloaded from

<div align="center">

http://mercurial.selenic.com/downloads/

</div>

There are also many other third-party clients which provides Hg services, for example: EasyMercurial[4] (cross platform) and SourceTree[5] (for Mac OS X only).

As a DVCS (Distributed Version Control System) source-control software, Hg is relatively new. But compared with other tools like Git, Hg is considered *friendlier* with a lower learning curve. This is despite the fact that Hg uses two distinct sets of commands and two distinct vocabularies for operations depending upon whether the repository is local or remote. Documentation for Hg is substantially better, including a book[6]. They've also had the advantage of trying the documentation

---

[1]Available online at http://fasp.sourceforge.net. It is also available in "faspsolver/doc/doc.zip".
[2]Official website: https://bitbucket.org/
[3]Official website: http://mercurial.selenic.com/
[4]Official website: http://easyhg.org
[5]Official website: http://www.sourcetreeapp.com
[6]The hgbook, http://hgbook.red-bean.com/

on a fairly savvy group of developers (Mozilla) who gave them lots of feedback that helped polish the rough edges.

Linux or Mac OS X

First, you need to obtain a free copy of FASP kernel functions from our public Hg repository. If you are downloading FASP for the first time, you can clone the repository to your local machine:

```
"Download FASP kernel subroutines via HTTPS"

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

If you have any problems when clone this repository, please send us an email to faspdev@gmail.com.

After a long pause[7], you should have obtained "*faspsolver*" in your current directory successfully. If you have already cloned the repository before, you can just pull a new version and update your local version with it: Go to your local "*faspsolver*" directory and then

```
"Pull a new version from BitBucket"
$ hg pull

"Update you local version to the new version"
$ hg update
```

Windows OS

If you are using Windows, you may want to install TortoiseHg[8]. After installing it, the TortoiseHg menu has been merged into the right-click menu of Windows Explore. You could download FASP copy from BitBucket.org. Choose "TortoiseHg" `-->` "Clone" in the pop-up menu, the source address is

```
https://faspusers@bitbucket.org/fasp/faspsovler
```

Then press "Clone" and you will obtain "faspsolver" in the directory you set.

---

[7]In fact, a very long pause. This is because the initial clone with copy all the history data which is about 400MB in total. Depending on the speed of your network, it could take 15 minutes to one hour.

[8]Official website: http://tortoisehg.bitbucket.org/

1.5   Licensing

This basic (kernel) FASP distribution is open-source and is licensed under GNU Lesser General Public License or LGPL. Other distributions may have different licensing (contact the developer team for details on this).

LICENSING: This software is free software distributed under the Lesser General Public License or LGPL, version 3.0 or any later versions. This software distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License http://www.gnu.org/licenses/ for more details.

1.6   Building and installing the FASP library and examples

FASP has been tested on Linux (Cent OS, Debian, Fedora, RedHat, Ubuntu), OS X (Leopard, Snow Leopard, Lion, Mavericks, El Capitan), and Windows (XP, Win 7) with a couple of compliers including GCC, G++, Clang, ICC, VC++, GFORTRAN, G95, IFORT.

Linux or OS X

Now we give a simple instruction on how to compile FASP on Linux or OS X: To build the FASP library, go to the "faspsolver" directory. First you need to modify the "*FASP.mk.example*" file according to your own need and save it as "FASP.mk". Then you can just type:

```
$ make config
$ make install
```

Note that, if you do not have "FASP.mk" present in the current directory, default settings will be used for building and installation FASP.

In order to make sure everything is OK, you can go to the "faspsolver/test" directory and try to run the test problem:

```
$ ./test.ex
```

If you need more help, you can use

```
$ make help
```

and you will get the following screen

```
================================================
||   Fast Auxiliary Space Preconditioners (FASP)   ||
================================================


Quick start:
To build FASP, copy "FASP.mk.example" to "FASP.mk" and put user-defined setting
there and then type "make config; make install".

More options:
================================================
$ make config          # Configure the building environment
$ make config CC=gcc   # Configure with a specific C compiler
$ make config debug=yes # Configure with compiler debug options ON
$ make config debug=all # Configure with FASP internal debug options ON
================================================
$ make                 # Compile the library (after "make config")
$ make install         # install FASP libraries and related files
$ make uninstall       # Remove installed files by "make install"
$ make headers         # Generate function decorations automatically
$ make docs            # Generate the FASP documentation with Doxygen
$ make clean           # Remove obj files but retain configuration options
$ make distclean       # Clean and completely removes the build directory
$ make version         # Show version information
$ make help            # Show this screen
================================================
```

To uninstall FASP and clean up the working directory, you can simply run

```
$ make uninstall
$ make distclean
```

Windows 7

We provide a Visual Studio 2008 (VS08) solution and a VS10 solution of FASP for Windows users. For example, you can just open "`faspsolver/vs08/faspsolver-vs08.sln`" if you are using VS08 as your default developing environment. Then a single-click at the "Build Solution" on the menu or "F7" key will give you all the FASP libraries and the test programs in "`faspsolver/test/`". The way for building in VS10 is similar.

You need a C/C++ complier and a Fortran compiler together with Visual Studio to build FASP. You can use either Microsoft Visual C++ or Intel C compiler, together with Intel Fortran compiler.

If you are using other versions of Visual Studio (like VS05 or VS12), do NOT convert the "VS08" solution file to your VS version because the FASP files might be cleaned up (removed) by Visual Studio automatically. You'd better create another solution to build all the libraries and test programs by yourselves.
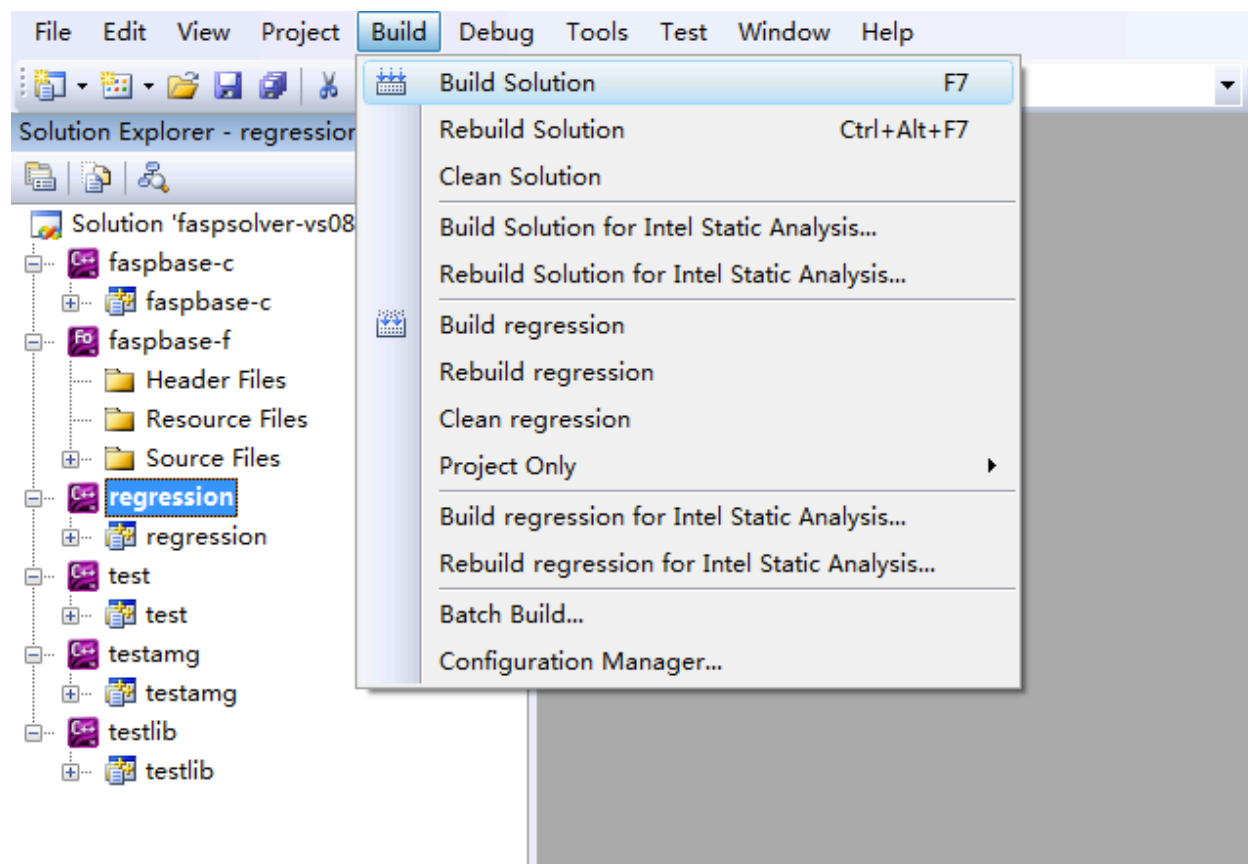
Figure 1.1: Build FASP using Visual Studio 2008.

If you need to build a VS solution by yourselves, you should create 5 projects:

1. "faspbase-c" contains all the ".c" and ".inl" files in the directory "./base/src/". You should add "./base/include" in Additional Directories. This project contains the core subroutines of faspsolver.

2. "faspbase-f" contains all the ".f" files in "./base/extra/sparsekit".

3. "testlib" contains all the ".c" files in "./test/src/". You should add "./test/include" in Additional Directories.

4. "test" is an executing program for test purpose in FASP. The source file is "./test/main/test.c".

5. "regression" is another executing program, which contains several methods to test the problems. The source file is "./test/main/regression.c".

NOTE: If you are using Visual C++, all the C files should be compiled as C++ code (by using the /TP compiling option).

After you successfully build the solution, you will get two static libraries named "faspbase-c-vs08.lib" and "faspbase-f-vs08.lib". You can use the "lib" command to wrap together as one single file (e.g. FASP.lib) for better portability. For example:

```
C:\FASP> lib /ltcg /out:FASP.lib faspbase-c-vs08.lib faspbase-f-vs08.lib
```

Using TCL based GUI

You can also try to build FASP using the simple TCL graphical user interface. For example, in Linux or Mac OS X, you may

```
$ wish FASP_install.tcl
```
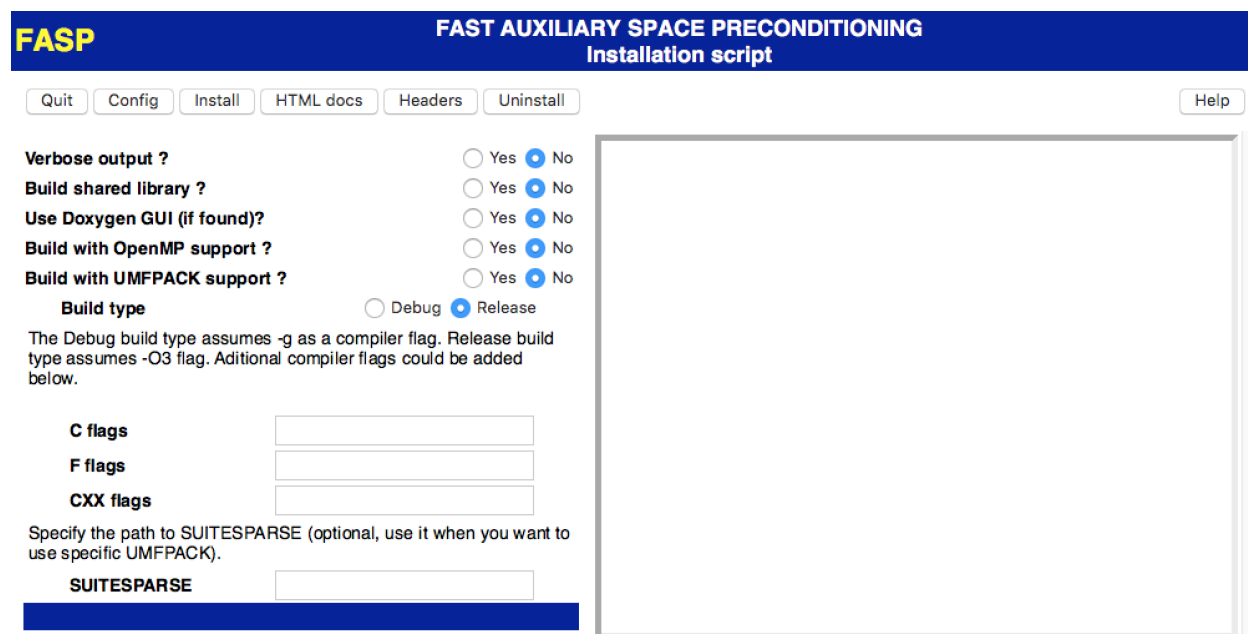

Figure 1.2: Install FASP using the TCL GUI on Mac OS X El Capitan.

A simple graphical interface will pop up; see Figure 1.2. The rest of the building process is straightforward: After choosing appropriate parameters, just click "Config" first and then followed by "Install".

External libraries

There are a few *optional* external libraries that you might want to use, including memory allocation routines, direct solvers, ILU methods, discretization packages, etc. FASP has interfaces to a couple of them which we often use, for example, UMFPack, SuperLU, MUMPS, SparseKit, dlmalloc.

Chapter 2

A Tutorial

In this chapter, we use a couple simple examples to demonstrate how to use the FASP package for solving existing linear systems which have been saved as disk files. All the examples can be found in "faspsolver/tutorial/". Here we only discuss the C version of these examples; interested users can read the F90 version of some of the examples. After you successfully build FASP (see §1.6), just go to the "faspsolver/tutorial/" directory and the compiled tutorial examples should be ready to be tried.

2.1   The first example

The first example is the simplest one that we can imagine: We read the stiffness matrix A and right-hand side b from disk files; then we solve $Ax = b$ using the classical AMG method [2, 13, 14]; see §3.7. The stiffness matrix A is symmetric positive definite (SPD), arising from the continuous piecewise linear finite element discretization of the Poisson equation

$$-\Delta u = f$$

(with the Dirichlet boundary condition) on a simple quasi-uniform triangulation of the bounded domain Ω.

```
1   /*! \file poisson-amg.c
2    *
3    *  \brief The first test example for FASP: using AMG to solve
4    *         the discrete Poisson equation from P1 finite element.
5    *         C version.
6    *
7    *  \note  AMG example for FASP: C version
8    *
9    *  Solving the Poisson equation (P1 FEM) with AMG
10   */
11
```

```c
12  #include "fasp.h"
13  #include "fasp_functs.h"
14
15  /**
16   * \fn int main (int argc, const char * argv[])
17   *
18   * \brief This is the main function for the first example.
19   *
20   * \author Chensong Zhang
21   * \date   12/21/2011
22   *
23   * Modified by Chensong Zhang on 09/22/2012
24   */
25  int main (int argc, const char * argv[])
26  {
27      input_param      inparam;  // parameters from input files
28      AMG_param        amgparam; // parameters for AMG
29
30      printf("\n=======================================");
31      printf("\n||   FASP: AMG example -- C version   ||");
32      printf("\n=======================================\n\n");
33
34      // Step 0. Set parameters: We can use ini/amg.dat
35      fasp_param_set(argc, argv, &inparam);
36      fasp_param_init(&inparam, NULL, &amgparam, NULL, NULL);
37
38      // Set local parameters using the input values
39      const int print_level = inparam.print_level;
40
41      // Step 1. Get stiffness matrix and right-hand side
42      // Read A and b -- P1 FE discretization for Poisson. The location
43      // of the data files is given in "ini/amg.dat".
44      dCSRmat A;
45      dvector b, x;
46      char filename1[512], *datafile1;
47      char filename2[512], *datafile2;
48
49      // Read the stiffness matrix from matFE.dat
50      strncpy(filename1,inparam.workdir,128);
51      datafile1="csrmat_FE.dat"; strcat(filename1,datafile1);
52
53      // Read the RHS from rhsFE.dat
54      strncpy(filename2,inparam.workdir,128);
55      datafile2="rhs_FE.dat"; strcat(filename2,datafile2);
56
57      fasp_dcsrvec2_read(filename1,filename2,&A,&b);
58
59      // Step 2. Print problem size and AMG parameters
60      if (print_level>PRINT_NONE) {
61          printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
62          printf("b: n = %d\n", b.row);
63          fasp_param_amg_print(&amgparam);
64      }
```

```
65
66        // Step 3. Solve the system with AMG as an iterative solver
67        // Set the initial guess to be zero and then solve it
68        // with AMG method as an iterative procedure
69        fasp_dvec_alloc ( A.row , &x );
70        fasp_dvec_set ( A.row , &x, 0.0 );
71
72        fasp_solver_amg (&A, &b, &x, &amgparam );
73
74        // Step 4. Clean up memory
75        fasp_dcsr_free (&A );
76        fasp_dvec_free (&b );
77        fasp_dvec_free (&x );
78
79        return FASP_SUCCESS ;
80    }
81
82    /*--------------------------------*/
83    /*--        End of File        --*/
84    /*--------------------------------*/
```

Since this is the first example, we will explain it in some detail:

- Line 1 tells the Doxygen documentation system that the filename is "poisson-amg.c". Line 3–5 tells the Doxygen what is the purpose of this file (function).

- Line 12–13 includes the main FASP header file "fasp.h" and FASP function decoration header "fasp_functs.h". These two headers shall be included in all files that requires FASP subroutines. Please also be noted that the function decorations in "fasp_functs.h" is automatically generated from the source files and should NOT be modified by an enduser.

- Line 35 sets solver parameters using the default parameters or from the command line options; see more discussions in §2.4. In the "tutorial/ini/amg.dat" file, we can set the location of the data files, type of solvers, maximal number of iteration numbers, convergence tolerance, and many other parameters for iterative solvers.

- Line 44 defines a sparse matrix A in the compressed sparse row (CSR) format. Line 45 defines two vectors: the right-hand side b and the numerical solution x. We refer to §3.1 for definitions of vectors and general sparse matrices.

- Line 57 reads the matrix and the right-hand side from two disk files. Line 49–58 defines the filenames of them.

- Line 60–64 prints basic information of coefficient matrix, right-hand side, and solver parameters.

- Line 69–70 allocates memory for the solution vector x and set its initial value to be all zero.

- Line 72 solves $Ax = b$ using the AMG method. Type of the AMG method and other parameters have been given in "amgparam" at Line 36; see §3.7.

- Line 75–77 frees up memory allocated for A, b, and x.

To run this example, we can simply type:

```
$ ./poisson−amg−c.ex
```

A sample output is given as follows (note that the actual output depends on the solver parameters and might be different than what you see here):

```
=============================================
||    FASP: AMG example −− C version    ||
=============================================

fasp_dcsrvec2_read: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec2_read: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969

        Parameters in AMG_param
----------------------------------------------------------------
AMG print level:                  2
AMG max num of iter:              1
AMG type:                         1
AMG tolerance:                    1.00e−06
AMG max levels:                   20
AMG cycle type:                   1
AMG coarse solver type:           0
AMG scaling of coarse correction: 0
AMG smoother type:                2
AMG smoother order:               1
AMG num of presmoothing:          1
AMG num of postsmoothing:         1
AMG coarsening type:              1
AMG interpolation type:           1
AMG dof on coarsest grid:         500
AMG strong threshold:             0.3000
AMG truncation threshold:         0.2000
AMG max row sum:                  0.9000
AMG aggressive levels:            0
AMG aggressive path:              1
----------------------------------------------------------------


Calling classical AMG ...
-----------------------------------------------------------------
  Level    Num of rows    Num of nonzeros    Avg. NNZ / row
-----------------------------------------------------------------
    0          3969            27281              6.87
```

```
      1              1985               28523            14.37
      2               541                7951            14.70
      3               141                1803            12.79
_____

  Grid complexity = 1.672   |   Operator complexity = 2.403
_____

Classical AMG setup costs 0.0068 seconds.
_____

It Num |    ||r||/||b||    |       ||r||      |  Conv. Factor
_____

      0 |  1.000000e+00   |   7.514358e+00   |     -.--
      1 |  9.851978e-03   |   7.403129e-02   |    0.0099
      2 |  3.507451e-04   |   2.635624e-03   |    0.0356
      3 |  1.764023e-05   |   1.325550e-04   |    0.0503
      4 |  8.820794e-07   |   6.628261e-06   |    0.0500
Number of iterations = 4 with relative residual 8.820794e-07.
AMG solve costs 0.0017 seconds.
AMG totally costs 0.0091 seconds.
```

We also provide a Fortran 90 example, which does the same thing as this C code except it gives less output, in "`tutorial/main/poisson-amg.f90`". Users who would like to call FASP from Fortran can learn from this simple example.

```fortran
 1  !> \file poisson-amg.f90
 2  !> \brief The first test example for FASP: using AMG to solve
 3  !>        the discrete Poisson equation from P1 finite element.
 4  !>        F90 version.
 5  !>
 6  !> \note  AMG example for FASP: F90 version
 7  !>
 8  !> Solving the Poisson equation (P1 FEM) with AMG
 9  !>
10  !> \author Chensong Zhang
11  !> \date    12/21/2011
12
13  program test
14
15    implicit none
16
17    double precision, dimension(:), allocatable :: u, b, a
18    integer,          dimension(:), allocatable :: ia, ja
19
20    integer          :: iufile, n, nnz, i, prt_lvl, maxit
21    double precision :: tol
22
23    print*, ""
24    write(*,"(A)") "========================================"
25    write(*,"(A)") "||   FASP: AMG example -- F90 version   ||"
26    write(*,"(A)") "========================================"
27    print*, ""
28
29    ! Step 0: user defined variables
```

```fortran
30      prt_lvl = 3
31      maxit = 100
32      tol = 1.0d-6
33      iufile = 1
34
35      ! Step 1: read A and b
36
37      !===> Read data A from file
38      open(unit=iufile,file='../data/csrmat_FE.dat')
39
40      read(iufile,*) n
41      allocate(ia(1:n+1))
42      read(iufile,*) (ia(i),i=1,n+1)
43
44      nnz=ia(n+1)-ia(1)
45      allocate(ja(1:nnz),a(1:nnz))
46      read(iufile,*) (ja(i),i=1,nnz)
47      read(iufile,*) (a(i),i=1,nnz)
48
49      close(iufile)
50
51      !===> Read data b from file
52      open(unit=iufile,file='../data/rhs_FE.dat')
53
54      read(iufile,*) n
55      allocate(b(1:n))
56      read(iufile,*) (b(i),i=1,n)
57
58      close(iufile)
59
60      !===> Shift the index to start from 0 (for C routines)
61      forall (i=1:n+1) ia(i)=ia(i)-1
62      forall (i=1:nnz) ja(i)=ja(i)-1
63
64      ! Step 2: Solve the system
65
66      !===> Initial guess
67      allocate(u(1:n))
68      u=0.0d0
69      call fasp_fwrapper_amg(n,nnz,ia,ja,a,b,u,tol,maxit,prt_lvl)
70
71      ! Step 3: Clean up memory
72      deallocate(ia,ja,a)
73      deallocate(b,u)
74
75  end program test
76
77  !/*--------------------------------*/
78  !/*--         End of File         --*/
79  !/*--------------------------------*/
```

2.2 The second example

In the second example, we modify the previous example slightly and solve the Poisson equation using iterative methods (here by default we use the Conjugate Gradient method without preconditioning).

```c
/*! \file poisson-its.c
 *  \brief The second test example for FASP: using ITS to solve
 *         the discrete Poisson equation from P1 finite element.
 *
 *  \note  ITS example for FASP: C version
 *
 *  Solving the Poisson equation (P1 FEM) with iterative methods
 */

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

/**
 * \fn int main (int argc, const char * argv[])
 *
 * \brief This is the main function for the second example.
 *
 * \author Feiteng Huang
 * \date   04/13/2012
 *
 * Modified by Chensong Zhang on 09/22/2012
 */
int main (int argc, const char * argv[])
{
    input_param         inparam;  // parameters from input files
    itsolver_param      itparam;  // parameters for itsolver

    printf("\n========================================");
    printf("\n||   FASP: ITS example -- C version    ||");
    printf("\n========================================\n\n");

    // Step 0. Set parameters: We can use ini/its.dat
    fasp_param_set(argc, argv, &inparam);
    fasp_param_init(&inparam, &itparam, NULL, NULL, NULL);

    // Set local parameters
    const int print_level = inparam.print_level;

    // Step 1. Get stiffness matrix and right-hand side
    // Read A and b -- P1 FE discretization for Poisson. The location
    // of the data files is given in "ini/its.dat".
    dCSRmat A;
    dvector b, x;
    char filename1[512], *datafile1;
    char filename2[512], *datafile2;

    // Read the stiffness matrix from matFE.dat
```

```c
48        strncpy ( filename1 , inparam . workdir ,128);
49        datafile1="csrmat_FE.dat"; strcat ( filename1 , datafile1 );
50
51        // Read the RHS from rhsFE.dat
52        strncpy ( filename2 , inparam . workdir ,128);
53        datafile2="rhs_FE.dat"; strcat ( filename2 , datafile2 );
54
55        fasp_dcsrvec2_read ( filename1 , filename2 ,&A,&b );
56
57        // Step 2. Print problem size and ITS parameters
58        if ( print_level >PRINT_NONE ) {
59            printf ("A: m = %d, n = %d, nnz = %d\n", A.row , A.col , A.nnz );
60            printf ("b: n = %d\n", b.row );
61            fasp_param_solver_print (&itparam );
62        }
63
64        // Step 3. Solve the system with ITS as an iterative solver
65        // Set the initial guess to be zero and then solve it using standard
66        // iterative methods , without applying any preconditioners
67        fasp_dvec_alloc (A.row, &x );
68        fasp_dvec_set (A.row ,&x ,0.0);
69
70        fasp_solver_dcsr_itsolver (&A, &b, &x, NULL , &itparam );
71
72        // Step 4. Clean up memory
73        fasp_dcsr_free (&A);
74        fasp_dvec_free (&b);
75        fasp_dvec_free (&x);
76
77        return FASP_SUCCESS ;
78  }
79
80  /*--------------------------------*/
81  /*--         End of File          --*/
82  /*--------------------------------*/
```

This example is very similar to the first example and we briefly explain the differences:

- Line 67–68 allocates memory for the solution vector x and set its initial value to be all zero.

- Line 70 solves $Ax = b$ using the general interface for Krylov subspace methods. Type the iterative method and other parameters have been specified in "itparam"; see §3.5 for details.

To run this example, we can simply type:

```
$ ./poisson−its−c.ex
```

A sample output is given as follows (note that the actual output depends on the solver parameters and might be different than what you see here):

```
==================================================
||     FASP: ITS example —— C version      ||
==================================================


fasp_dcsrvec2_read: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec2_read: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969


        Parameters in itsolver_param
────────────────────────────────────────────────
Solver print level:            2
Solver type:                   1
Solver precond type:           2
Solver max num of iter:        500
Solver tolerance:              1.00e−06
Solver stopping type:          1
────────────────────────────────────────────────



Calling PCG solver (CSR) ...
──────────────────────────────────────────────────────
It Num |    ||r||/||b||    |       ||r||       |  Conv. Factor
──────────────────────────────────────────────────────
      0 |   1.000000e+00    |   7.514358e+00    |     −.——
      1 |   5.078029e−01    |   3.815813e+00    |    0.5078
      2 |   3.728856e−01    |   2.801996e+00    |    0.7343
      3 |   3.359470e−01    |   2.524426e+00    |    0.9009
      4 |   2.590574e−01    |   1.946650e+00    |    0.7711
      5 |   2.380797e−01    |   1.789016e+00    |    0.9190
      6 |   1.992579e−01    |   1.497295e+00    |    0.8369
      7 |   1.847971e−01    |   1.388631e+00    |    0.9274
      8 |   1.619777e−01    |   1.217158e+00    |    0.8765
      9 |   1.513446e−01    |   1.137257e+00    |    0.9344
     10 |   1.364935e−01    |   1.025661e+00    |    0.9019
     11 |   1.283425e−01    |   9.644117e−01    |    0.9403
     12 |   1.179652e−01    |   8.864327e−01    |    0.9191
     13 |   1.115146e−01    |   8.379605e−01    |    0.9453
     14 |   1.038726e−01    |   7.805360e−01    |    0.9315
     15 |   9.863412e−02    |   7.411721e−01    |    0.9496
     16 |   9.277360e−02    |   6.971341e−01    |    0.9406
     17 |   8.842679e−02    |   6.644706e−01    |    0.9531
     18 |   8.378399e−02    |   6.295829e−01    |    0.9475
     19 |   8.011023e−02    |   6.019770e−01    |    0.9562
     20 |   7.633221e−02    |   5.735875e−01    |    0.9528
     21 |   7.317756e−02    |   5.498824e−01    |    0.9587
     22 |   7.003292e−02    |   5.262524e−01    |    0.9570
     23 |   6.728610e−02    |   5.056119e−01    |    0.9608
     24 |   6.461736e−02    |   4.855580e−01    |    0.9603
     25 |   6.219614e−02    |   4.673640e−01    |    0.9625
     26 |   5.989276e−02    |   4.500557e−01    |    0.9630
     27 |   5.773520e−02    |   4.338429e−01    |    0.9640
     28 |   5.571758e−02    |   4.186818e−01    |    0.9651
```

```
29  |    5.377630e-02    |    4.040944e-01    |      0.9652
30  |    5.198586e-02    |    3.906404e-01    |      0.9667
31  |    5.022413e-02    |    3.774021e-01    |      0.9661
32  |    4.861699e-02    |    3.653255e-01    |      0.9680
33  |    4.700598e-02    |    3.532197e-01    |      0.9669
34  |    4.554874e-02    |    3.422696e-01    |      0.9690
35  |    4.406559e-02    |    3.311246e-01    |      0.9674
36  |    4.273253e-02    |    3.211075e-01    |      0.9697
37  |    4.135901e-02    |    3.107864e-01    |      0.9679
38  |    4.013076e-02    |    3.015569e-01    |      0.9703
39  |    3.885861e-02    |    2.919975e-01    |      0.9683
40  |    3.776252e-02    |    2.837611e-01    |      0.9718
41  |    3.678565e-02    |    2.764205e-01    |      0.9741
42  |    3.648645e-02    |    2.741722e-01    |      0.9919
43  |    3.725368e-02    |    2.799375e-01    |      1.0210
44  |    3.922957e-02    |    2.947850e-01    |      1.0530
45  |    4.003513e-02    |    3.008383e-01    |      1.0205
46  |    3.683219e-02    |    2.767703e-01    |      0.9200
47  |    3.161285e-02    |    2.375503e-01    |      0.8583
48  |    2.944107e-02    |    2.212307e-01    |      0.9313
49  |    2.961834e-02    |    2.225628e-01    |      1.0060
50  |    2.774118e-02    |    2.084571e-01    |      0.9366
51  |    2.513603e-02    |    1.888811e-01    |      0.9061
52  |    2.489908e-02    |    1.871006e-01    |      0.9906
53  |    2.379644e-02    |    1.788150e-01    |      0.9557
54  |    2.190590e-02    |    1.646088e-01    |      0.9206
```

2.3   The third example

This example is slightly longer and is a modification of the previous one. In this example, we wish to demonstrate how to setup a simple preconditioner for the preconditioned conjugate gradient (PCG) method.

```
1   /*! \file poisson-pcg.c
2    *  \brief The third test example for FASP: using PCG to solve
3    *         the discrete Poisson equation from P1 finite element.
4    *         C version.
5    *
6    *  \note  PCG example for FASP: C version
7    *
8    *  Solving the Poisson equation (P1 FEM) with PCG methods
9    */
10
11  #include "fasp.h"
12  #include "fasp_functs.h"
13
14  /**
15   * \fn int main (int argc, const char * argv[])
16   *
17   * \brief This is the main function for the third example.
```

```
18    *
19    * \author Feiteng Huang
20    * \date   05/17/2012
21    *
22    * Modified by Chensong Zhang on 09/22/2012
23    */
24   int main (int argc, const char * argv[])
25   {
26       input_param          inparam;  // parameters from input files
27       itsolver_param       itparam;  // parameters for itsolver
28       AMG_param            amgparam; // parameters for AMG
29       ILU_param            iluparam; // parameters for ILU
30
31       printf("\n=========================================");
32       printf("\n||   FASP: PCG example -- C version   ||");
33       printf("\n=========================================\n\n");
34
35       // Step 0. Set parameters: We can use ini/pcg.dat
36       fasp_param_set(argc, argv, &inparam);
37       fasp_param_init(&inparam, &itparam, &amgparam, &iluparam, NULL);
38
39       // Set local parameters
40       const SHORT print_level = itparam.print_level;
41       const SHORT pc_type     = itparam.precond_type;
42       const SHORT stop_type   = itparam.stop_type;
43       const INT   maxit       = itparam.maxit;
44       const REAL  tol         = itparam.tol;
45
46       // Step 1. Get stiffness matrix and right-hand side
47       // Read A and b -- P1 FE discretization for Poisson. The location
48       // of the data files is given in "ini/pcg.dat".
49       dCSRmat A;
50       dvector b, x;
51       char filename1[512], *datafile1;
52       char filename2[512], *datafile2;
53
54       // Read the stiffness matrix from matFE.dat
55       strncpy(filename1,inparam.workdir,128);
56       datafile1="csrmat_FE.dat"; strcat(filename1,datafile1);
57
58       // Read the RHS from rhsFE.dat
59       strncpy(filename2,inparam.workdir,128);
60       datafile2="rhs_FE.dat"; strcat(filename2,datafile2);
61
62       fasp_dcsrvec2_read(filename1,filename2,&A,&b);
63
64       // Step 2. Print problem size and PCG parameters
65       if (print_level>PRINT_NONE) {
66           printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
67           printf("b: n = %d\n", b.row);
68           fasp_param_solver_print(&itparam);
69       }
70
```

```
71        // Setp 3. Setup preconditioner
72        // Preconditioner type is determined by pc_type
73        precond *pc = fasp_precond_setup(pc_type, &amgparam, &iluparam, &A);
74
75        // Step 4. Solve the system with PCG as an iterative solver
76        // Set the initial guess to be zero and then solve it using PCG solver
77        // Note that we call PCG interface directly. There is another way which
78        // calls the abstract iterative method interface; see possion-its.c for
79        // more details.
80        fasp_dvec_alloc(A.row, &x);
81        fasp_dvec_set(A.row, &x, 0.0);
82
83        fasp_solver_dcsr_pcg(&A, &b, &x, pc, tol, maxit, stop_type, print_level);
84
85        // Step 5. Clean up memory
86        if (pc_type!=PREC_NULL) fasp_mem_free(pc->data);
87        fasp_dcsr_free(&A);
88        fasp_dvec_free(&b);
89        fasp_dvec_free(&x);
90
91        return FASP_SUCCESS;
92    }
93
94    /*---------------------------------*/
95    /*--        End of File         --*/
96    /*---------------------------------*/
```

This example is very similar to the first example and we now briefly explain it:

- Line 36 sets default parameters. In this example, we need parameters for iterative methods, AMG preconditioner, and ILU preconditioner.

- Line 73 sets up the desired preconditioner and prepare it for the preconditioned iterative methods.

- Line 83 calls PCG to solve $Ax = b$. One can also call the general iterative method interface as in the previous example.

- Line 86 cleans up auxiliary data associated with the preconditioner in use if necessary.

To run this example, we can simply type:

```
$ ./poisson-pcg-c.ex
```

A sample output is given as follows (note that the actual output depends on the solver parameters and might be different than what you see here):

```
===================================================
||    FASP: PCG example -- C version    ||
```

```
════════════════════════════════════

fasp_dcsrvec2_read: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec2_read: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969

        Parameters in itsolver_param
─────────────────────────────────────────────────────
Solver print level:            2
Solver type:                   1
Solver precond type:           2
Solver max num of iter:        500
Solver tolerance:              1.00e−06
Solver stopping type:          1
─────────────────────────────────────────────────────


─────────────────────────────────────────────────────
  Level    Num of rows    Num of nonzeros    Avg. NNZ / row
─────────────────────────────────────────────────────
    0           3969           27281            6.87
    1           1985           28523           14.37
    2            541            7951           14.70
    3            141            1803           12.79
─────────────────────────────────────────────────────
  Grid complexity = 1.672   |  Operator complexity = 2.403
─────────────────────────────────────────────────────
Classical AMG setup costs 0.0042 seconds.
─────────────────────────────────────────────────────
It Num |   ||r||/||b||    |      ||r||       |  Conv. Factor
─────────────────────────────────────────────────────
    0 |  1.000000e+00   |  7.514358e+00  |     −.−−
    1 |  1.156153e−02   |  8.687750e−02  |     0.0116
    2 |  3.127181e−04   |  2.349876e−03  |     0.0270
    3 |  4.813471e−06   |  3.617014e−05  |     0.0154
    4 |  5.312526e−08   |  3.992022e−07  |     0.0110
Number of iterations = 4 with relative residual 5.312526e−08.
```

2.4   Set parameters

In the previous examples, we have seen how to set solver parameters using default setting. Now we give an example on reading input parameters from a disk file like "tutorial/ini/amg.dat".

```
$ ./poisson−amg−c.ex −ini ini/amg.dat
```

We take "tutorial/ini/amg.dat" as an example:

```
1  %─────────────────────────────────────────────%
2  % input parameters                             %
```

```
3   % lines starting with % are comments          %
4   % must have spaces around the equal sign "="   %
5   %—————————————————————————%
6
7   workdir = ../data/     % work directory, no more than 128 characters
8   print_level = 3        % How much information to print out
9
10  %—————————————————————————%
11  % parameters for multilevel iteration          %
12  %—————————————————————————%
13
14  AMG_type                = SA      % C classic AMG
15                                    % SA smoothed aggregation
16                                    % UA unsmoothed aggregation
17  AMG_cycle_type          = V       % V V−cycle | W W−cycle
18                                    % A AMLI−cycle | NA Nonlinear AMLI−cycleA
19  AMG_tol                 = 1e−8    % tolerance for AMG
20  AMG_maxit               = 100     % number of AMG iterations
21  AMG_levels              = 20      % max number of levels
22  AMG_coarse_dof          = 500     % max number of coarse degrees of freedom
23  AMG_coarse_scaling      = OFF     % switch of scaling of the coarse grid correction
24  AMG_amli_degree         = 2       % degree of the polynomial used by AMLI cycle
25  AMG_nl_amli_krylov_type = 6       % Krylov method in NLAMLI cycle: 6 FGMRES | 7 GCG
26
27  %—————————————————————————%
28  % parameters for AMG smoothing                 %
29  %—————————————————————————%
30
31  AMG_smoother            = GS      % GS | JACOBI | SGS
32                                    % SOR | SSOR | GSOR | SGSOR | POLY
33  AMG_ILU_levels          = 0       % number of levels using ILU smoother
34  AMG_Schwarz_levels      = 0       % number of levels using Schwarz smoother
35  AMG_relaxation          = 1.1     % relaxation parameter for SOR smoother
36  AMG_polynomial_degree   = 3       % degree of the polynomial smoother
37  AMG_presmooth_iter      = 2       % number of presmoothing sweeps
38  AMG_postsmooth_iter     = 2       % number of postsmoothing sweeps
39
40  %—————————————————————————%
41  % parameters for classical AMG SETUP           %
42  %—————————————————————————%
43
44  AMG_coarsening_type     = 1       % 1 Modified RS
45                                    % 3 Compatible Relaxation
46                                    % 4 Aggressive
47  AMG_interpolation_type  = 1       % 1 Direct | 2 Standard | 3 Energy−min
48  AMG_strong_threshold    = 0.6     % Strong threshold
49  AMG_truncation_threshold = 0.4    % Truncation threshold
50  AMG_max_row_sum         = 0.9     % Max row sum
51
52  %—————————————————————————%
53  % parameters for aggregation−type AMG SETUP    %
54  %—————————————————————————%
55
```

```
56  AMG_strong_coupled        = 0.08   % Strong coupled threshold
57  AMG_max_aggregation       = 20     % Max size of aggregations
58  AMG_tentative_smooth      = 0.67   % Smoothing factor for tentative prolongation
59  AMG_smooth_filter         = OFF    % Switch for filtered matrix for smoothing
```

We now briefly discuss the parameters above: This example is very similar to the first example and we now briefly explain it:

- Line 7 sets the working directory, which should contain data files for the matrices (and right-hand side vectors when necessary).

- Line 8 sets the level of output for FASP routines. It should range from 0 to 10 with 0 means no output and 10 means output everything possible.

- Line 14–25 sets the basic parameters for multilevel iterations. For example, type of AMG, type of multilevel cycles, number of maximal levels, etc.

- Line 31–38 sets the type of smoothers, number of smoothing sweeps, etc.

- Line 44–50 sets the parameters for the setup phase of the classical AMG method (§3.7).

- Line 56–59 gives the parameters for the setup phase of the aggregation-base AMG methods (§3.7).

You can do a very simple experiment and change the AMG type from the classical AMG to smoothed aggregation AMG by revise Line 14 to

```
AMG_type                      = SA
```

Then you run "poisson-amg-c.ex" one more time and will get

```
=================================================
||    FASP: AMG example —— C version    ||
=================================================

fasp_dcsrvec2_read: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec2_read: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969


        Parameters in AMG_param
_____
AMG print level:              3
AMG max num of iter:          100
AMG type:                     2
AMG tolerance:                1.00e−08
AMG max levels:               20
AMG cycle type:               1
```

```
AMG coarse solver type:                 0
AMG scaling of coarse correction:       0
AMG smoother type:                      2
AMG smoother order:                     1
AMG num of presmoothing:                2
AMG num of postsmoothing:               2
Aggregation type:                       1
Aggregation number of pairs:            2
Aggregation quality bound:              8.00
_____


Calling SA AMG ...
_____
   Level     Num of rows    Num of nonzeros    Avg. NNZ / row
_____
     0           3969             27281            6.87
     1            541              6531           12.07
     2             41               421           10.27
_____
  Grid complexity = 1.147  |  Operator complexity = 1.255
_____
Smoothed aggregation setup costs 0.0028 seconds.
_____
It Num |    ||r||/||b||    |       ||r||      |  Conv. Factor
_____
     0 |   1.000000e+00    |   7.514358e+00   |     -.--
     1 |   4.345463e-02    |   3.265336e-01   |     0.0435
     2 |   8.041967e-03    |   6.043022e-02   |     0.1851
     3 |   3.808810e-03    |   2.862076e-02   |     0.4736
     4 |   1.838990e-03    |   1.381883e-02   |     0.4828
     5 |   8.675952e-04    |   6.519421e-03   |     0.4718
     6 |   4.089274e-04    |   3.072827e-03   |     0.4713
     7 |   1.939823e-04    |   1.457653e-03   |     0.4744
     8 |   9.276723e-05    |   6.970862e-04   |     0.4782
     9 |   4.471799e-05    |   3.360270e-04   |     0.4820
    10 |   2.171249e-05    |   1.631554e-04   |     0.4855
    11 |   1.060934e-05    |   7.972239e-05   |     0.4886
    12 |   5.212246e-06    |   3.916668e-05   |     0.4913
    13 |   2.572464e-06    |   1.933042e-05   |     0.4935
    14 |   1.274466e-06    |   9.576797e-06   |     0.4954
    15 |   6.333891e-07    |   4.759512e-06   |     0.4970
    16 |   3.155926e-07    |   2.371476e-06   |     0.4983
    17 |   1.575755e-07    |   1.184079e-06   |     0.4993
    18 |   7.881043e-08    |   5.922098e-07   |     0.5001
    19 |   3.947044e-08    |   2.965950e-07   |     0.5008
    20 |   1.978978e-08    |   1.487075e-07   |     0.5014
    21 |   9.931176e-09    |   7.462641e-08   |     0.5018
Number of iterations = 21 with relative residual 9.931176e-09.
AMG solve costs 0.0053 seconds.
AMG totally costs 0.0083 seconds.
```

You can compare this with the sample results in §2.1.

The input parameters allowed in FASP are not limited to the ones listed in this example. A list of possible iterative methods and preconditioners can be found in "base/include/fasp_const.h"; see §4.2. For more parameters and their ranges, we refer to the FASP Reference Manual.

Using "-ini [FILE]" is just one example of allowed command line option. To find out more what command line options are acceptable, you can type in a terminal window:

```
$ ./poisson−amg−c.ex −help
```

which will give you something like

```
=========================================
||    FASP: AMG example −− C version    ||
=========================================

FASP command line options:
=========================================================================
  −ini              [CharValue]  : Ini file name
  −print            [IntValue]   : Print level
  −output           [IntValue]   : Output to screen or a log file
  −solver           [IntValue]   : Solver type
  −precond          [IntValue]   : Preconditioner type
  −maxit            [IntValue]   : Max number of iterations
  −tol              [RealValue]  : Tolerance for iterative solvers
  −amgmaxit         [IntValue]   : Max number of AMG iterations
  −amgtol           [RealValue]  : Tolerance for AMG methods
  −amgtype          [IntValue]   : AMG type
  −amgcycle         [IntValue]   : AMG cycle type
  −amgcoarsening    [IntValue]   : AMG coarsening type
  −amginterpolation [IntValue]   : AMG interpolation type
  −amgsmoother      [IntValue]   : AMG smoother type
  −amgsthreshold    [RealValue]  : AMG strong threshold
  −amgscoupled      [RealValue]  : AMG strong coupled threshold
  −help                          : Brief help messages
```

Chapter 3

Basic Usage

In this chapter, we discuss the basic data structures and important building blocks which will be useful later for constructing auxiliary space preconditioners for systems of PDEs in Chapter 4. In particular, we will discuss vectors, sparse matrices, iterative methods, and multigrid methods.

3.1   Vectors and sparse matrices

The most important data structures for iterative methods are probably vectors and sparse matrices. In this section, we first discuss the data structures for vectors and matrices in FASP; and then we discuss BLAS for sparse matrices. The definitions can be found in "`base/include/fasp.h`".

Vectors

The data structure for vectors is very simple. It only contains the length of the vector and an array which contains the entries of this vector.

```
330   /**
331    * \struct dvector
332    * \brief Vector with n entries of REAL type
333    */
334   typedef struct dvector{
335
336       //! number of rows
337       INT row;
338
339       //! actual vector entries
340       REAL *val;
341
342   } dvector; /**< Vector of REAL type */
```

Sparse matrices

On the other hand, sparse matrices for PDE applications are very complicated. It depends on the particular applications, discretization methods, as well as solution algorithms. In FASP, there are several types of sparse matrices, COO, CSR, CSRL, BSR, and CSR Block, etc. The presentation closely follows ideas from Pissanetzky [12].

In this section, we use the following sparse matrix as an example to explain different formats for sparse matrices:

Example 3.1.1 *Consider the following* 4×5 *matrix with 12 non-zero entries*

$$\begin{pmatrix} 1 & 1.5 & 0 & 0 & 12 \\ 0 & 1 & 6 & 7 & 1 \\ 3 & 0 & 6 & 0 & 0 \\ 1 & 0 & 2 & 0 & 5 \end{pmatrix}$$

(i) COO format

The coordinate (COO) format or IJ format is the simplest sparse matrix format.

```
192  /**
193   * \struct dCOOmat
194   * \brief Sparse matrix of REAL type in COO (or IJ) format
195   *
196   * Coordinate Format (I,J,A)
197   *
198   * \note The starting index of A is 0.
199   * \note Change I to rowind, J to colind. To avoid with complex.h confliction on I.
200   */
201  typedef struct dCOOmat{
202
203      //! row number of matrix A, m
204      INT row;
205
206      //! column of matrix A, n
207      INT col;
208
209      //! number of nonzero entries
210      INT nnz;
211
212      //! integer array of row indices, the size is nnz
213      INT *rowind;
214
215      //! integer array of column indices, the size is nnz
216      INT *colind;
217
218      //! nonzero entries of A
219      REAL *val;
```

```
220
221  } dCOOmat; /**< Sparse matrix of REAL type in COO format */
```

So it clear that the sparse matrix in Example 3.1.1 in COO format is stored as:

```
row = 4
col = 5
nnz = 12

 I J  val
_____
 0 0   1.0
 0 1   1.5
 0 4  12.0
 1 1   1.0
 1 2   6.0
 1 3   7.0
 1 4   1.0
 .......
```

Although the COO format is easy to understand or use, it wastes storage space and has little advantages in sparse BLAS operations.

NOTE: In FASP, the indices always start from 0, instead of from 1. This is often the source of problems related to vectors and matrices.

(ii) CSR format

The most commonly used data structure for sparse matrices nowadays is probably the so-called *compressed sparse row* (CSR) format, according to Saad [15]. The compressed row storage format of a matrix $A \in \mathbb{R}^{n \times m}$ (n rows and m columns) consists of three arrays, as follows:

1. An integer array of row pointers of size n+1;

2. An integer array of column indexes of size nnz;

3. An array of actual matrix entries.

In FASP, we define:

```
132  /**
133   * \struct dCSRmat
134   * \brief Sparse matrix of REAL type in CSR format
135   *
136   * CSR Format (IA,JA,A) in REAL
137   *
138   * \note The starting index of A is 0.
```

```
139    */
140  typedef struct dCSRmat{
141
142      //! row number of matrix A, m
143      INT row;
144
145      //! column of matrix A, n
146      INT col;
147
148      //! number of nonzero entries
149      INT nnz;
150
151      //! integer array of row pointers, the size is m+1
152      INT *IA;
153
154      //! integer array of column indexes, the size is nnz
155      INT *JA;
156
157      //! nonzero entries of A
158      REAL *val;
159
160  } dCSRmat; /**< Sparse matrix of REAL type in CSR format */
```

The matrix (only nonzero elements) is stored in the array val row after row, in a way that i-th row begins at $val(IA(i))$ and ends at $val(IA(i + 1) - 1)$. In the same way, $JA(IA(i)$ to $JA(IA(i + 1) - 1)$ will contain the column indexes of the non-zeros in row i. Thus IA is of size $n + 1$ (number of rows in val plus one), JA and val are of size equal to the number of non-zeroes. The total number of non-zeroes is equal to $IA(n + 1) - 1$.

NOTE: When the sparse matrix A is a boolean (i.e. all entries are either 0 or 1), the actual non-zeroes are not stored because it is understood that, if it is nonzero, it could only be 1 and there is no need to store it.

The matrix in Example 3.1.1 in CSR format is represented in the following way:

- IA is of size 5 and
$$IA = \left\| \, 0 \, \right\| \, 3 \, \left\| \, 7 \, \right\| \, 9 \, \left\| \, 12 \, \right\|$$

- JA is of size $IA(5) - 1 = 12$
$$JA = \left\| \, 0 \, \right| \, 1 \, \left| \, 4 \, \right\| \, 1 \, \left| \, 3 \, \right| \, 2 \, \left| \, 4 \, \right\| \, 0 \, \left| \, 2 \, \right\| \, 2 \, \left| \, 4 \, \right| \, 0 \, \right\|$$

- val is of the same size as JA and
$$val = \left\| \, 1. \, \right| \, 1.5 \, \left| \, 12. \, \right\| \, 1. \, \left| \, 7. \, \right| \, 6. \, \left| \, 1. \, \right\| \, 3. \, \left| \, 6. \, \right\| \, 2. \, \left| \, 5. \, \right| \, 1. \, \right\|$$

Here we use double vertical bars to separate rows and single vertical bars to separate values.

Below is a "non-numeric" example.

Example 3.1.2 *Consider the following sparse matrix:*

$$\begin{pmatrix} a_{11} & 0 & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & a_{32} & 0 & a_{34} \\ a_{41} & a_{42} & a_{43} & 0 \end{pmatrix}$$

For this matrix, we have that the number of non-zeros nnz $= 10$. Furthermore, the three arrays of in the CSR format are:

$$IA = \left\| \; 0 \; \right\| \; 2 \; \left\| \; 5 \; \right\| \; 7 \; \left\|, \right.$$

$$JA = \left\| \; 0 \; \right| \; 2 \; \left\| \; 1 \; \right| \; 2 \; \left| \; 3 \; \right\| \; 1 \; \left| \; 3 \; \right\| \; 0 \; \left| \; 1 \; \right| \; 2 \; \left\|, \right.$$

and

$$val = \left\| \; a_{11} \; \right| \; a_{13} \; \left\| \; a_{22} \; \right| \; a_{23} \; \left| \; a_{24} \; \right\| \; a_{32} \; \left| \; a_{34} \; \right\| \; a_{41} \; \left| \; a_{42} \; \right| \; a_{43} \; \left\|. \right.$$

(iii) CSRL format

CSRL matrix format [10] groups rows with same number of nonzeros together and improves cache hitting rate.

```
253  /*!
254   * \struct dCSRLmat
255   * \brief Sparse matrix of REAL type in CSRL format
256   */
257  typedef struct dCSRLmat{
258
259      //! number of rows
260      INT row;
261
262      //! number of cols
263      INT col;
264
265      //! number of nonzero entries
266      INT nnz;
```

```
267
268        //! number of different values in i-th row, i=0:nrows-1
269        INT dif;
270
271        //! nz_diff[i]: the i-th different value in 'nzrow'
272        INT *nz_diff;
273
274        //! row index of the matrix (length-grouped): rows with same nnz are together
275        INT *index;
276
277        //! j in {start[i],...,start[i+1]-1} means nz_diff[i] nnz in index[j]-row
278        INT *start;
279
280        //! column indices of all the nonzeros
281        INT *ja;
282
283        //! values of all the nonzero entries
284        REAL *val;
285
286 } dCSRLmat; /**< Sparse matrix of REAL type in CSRL format */
```

3.2   Block sparse matrices

For PDE applications, we often need to solve systems of partial differential equations. Many iterative methods and preconditioners could take advantages of the structure of PDE systems and improve efficiency. So we often need to use semi-structured (block) sparse data structures to store the coefficient matrix arising from PDE systems.

Depending on different applications and different solving algorithms, we can use two types of block matrices: dBSRmat (or BSR Block Compressed Sparse Row) and block_dCSRmat (CSR Block or Block of CSR matrices).

For more details as well as other specialized block matrices, readers are referred to the header file "base/include/fasp_block.h".

As an example, we consider the following matrix, which have been used in §3.1 for the CSR format. We add structure to this matrix and divide it as a 2×2 block matrix:

Example 3.2.1

$$
\left(
\begin{array}{cc|cc}
a_{11} & 0 & a_{13} & 0 \\
0 & a_{22} & a_{23} & a_{24} \\
\hline
0 & a_{32} & 0 & a_{34} \\
a_{41} & a_{42} & a_{43} & 0
\end{array}
\right)
$$

(i) BSR format

This format is a standard data structure for storing block sparse matrices which has been used by
the Intel MKL library.

```
33  /**
34   * \struct dBSRmat
35   * \brief Block sparse row storage matrix of REAL type
36   *
37   * \note This data structure is adapted from the Intel MKL library. Refer to:
38   * http://software.intel.com/sites/products/documentation/hpc/mkl/lin/index.htm
39   *
40   * \note Some of the following entries are capitalized to stress that they are
41   *       for blocks!
42   */
43  typedef struct dBSRmat {
44
45      //! number of rows of sub-blocks in matrix A, M
46      INT ROW;
47
48      //! number of cols of sub-blocks in matrix A, N
49      INT COL;
50
51      //! number of nonzero sub-blocks in matrix A, NNZ
52      INT NNZ;
53
54      //! dimension of each sub-block
55      INT nb; // NOTE: for the moment, allow nb*nb full block
56
57      //! storage manner for each sub-block
58      INT storage_manner; // 0: row-major order, 1: column-major order
59
60      //! A real array that contains the elements of the non-zero blocks of
61      //! a sparse matrix. The elements are stored block-by-block in row major
62      //! order. A non-zero block is the block that contains at least one non-zero
63      //! element. All elements of non-zero blocks are stored, even if some of
64      //! them is equal to zero. Within each nonzero block elements are stored
65      //! in row-major order and the size is (NNZ*nb*nb).
66      REAL *val;
67
68      //! integer array of row pointers, the size is ROW+1
69      INT *IA;
70
71      //! Element i of the integer array columns is the number of the column in the
72      //! block matrix that contains the i-th non-zero block. The size is NNZ.
73      INT *JA;
74
75  } dBSRmat; /**< Matrix of REAL type in BSR format */
```

For the matrix in Example 3.2.1, we have that the number of block rows $ROW = 2$, the number
of block columns $COL = 2$, and the number of block nonzeros $NNZ = 4$. The block size is $nb = 2$.

We can choose different storage manners for storing the small blocks. Suppose that we set it to be 0, i.e. row-major format. Then the three arrays of in the BSR format are:

$$IA = \left\| \, 0 \, \right\| \, 8 \, \left\| \, 16 \, \right\|,$$

$$JA = \left\| \, 0 \, \right| \, 1 \, \left\| \, 0 \, \right| \, 1 \, \left\|,$$

and

$$val = \left\| \, a_{11} \, \right| \, 0 \, \left| \, 0 \, \right| \, a_{22} \, \left\| \, a_{13} \, \right| \, 0 \, \left| \, a_{23} \, \right| \, a_{24} \, \right\|$$
$$\left\| \, 0 \, \right| \, a_{32} \, \left| \, a_{41} \, \right| \, a_{42} \, \left\| \, 0 \, \right| \, a_{34} \, \left| \, a_{43} \, \right| \, 0 \, \right\|.$$

We immediately notice that this format might be not be the best choice for this particular matrix due to all the blocks are nonzero blocks, i.e., contain nonzero entries. However, for PDE applications, this does not usually happen.

(ii) CSR Block format

This format is simple and is derived from the dCSRmat data structure. The following definition explains itself.

```
77   /**
78    * \struct block_dCSRmat
79    * \brief Block REAL CSR matrix format
80    *
81    * \note The starting index of A is 0.
82    */
83   typedef struct block_dCSRmat {
84
85       //! row number of blocks in A, m
86       INT brow;
87
88       //! column number of blocks A, n
89       INT bcol;
90
91       //! blocks of dCSRmat, point to blocks[brow][bcol]
92       dCSRmat **blocks;
93
94   } block_dCSRmat; /**< Matrix of REAL type in Block CSR format */
```

3.3   I/O subroutines for sparse matrices

In FASP, we provided several functions for reading, writing, and printing different formats of sparse matrices in plain text or binary formats. These functions can be found in "base/src/io.c" and we list the available functions as follows:

```
834   /*-------- In file: io.c --------*/
835
836   void fasp_dcsrvec1_read (const char *filename,
837                                  dCSRmat *A,
838                                  dvector *b);
839
840   void fasp_dcsrvec2_read (const char *filemat,
841                                  const char *filerhs,
842                                  dCSRmat *A,
843                                  dvector *b );
844
845   void fasp_dcsr_read (const char *filename,
846                              dCSRmat *A);
847
848   void fasp_dcoo_read (const char *filename,
849                              dCSRmat *A);
850
851   void fasp_dcoo1_read (const char *filename,
852                               dCOOmat *A);
853
854   void fasp_dcoo_shift_read (const char *filename,
855                                    dCSRmat *A);
856
857   void fasp_dmtx_read (const char *filename,
858                              dCSRmat *A);
859
860   void fasp_dmtxsym_read (const char *filename,
861                                 dCSRmat *A);
862
863   void fasp_dstr_read (const char *filename,
864                              dSTRmat *A);
865
866   void fasp_dbsr_read (const char *filename,
867                              dBSRmat *A);
868
869   void fasp_dvecind_read (const char *filename,
870                                 dvector *b);
871
872   void fasp_dvec_read (const char *filename,
873                              dvector *b);
874
875   void fasp_ivecind_read (const char *filename,
876                                 ivector *b);
877
878   void fasp_ivec_read (const char *filename,
879                              ivector *b);
880
881   void fasp_dcsrvec1_write (const char *filename,
882                                   dCSRmat *A,
883                                   dvector *b);
884
885   void fasp_dcsrvec2_write (const char *filemat,
886                                   const char *filerhs,
```

```
887                              dCSRmat  *A,
888                              dvector  *b  );
889
890  void fasp_dcoo_write (const char *filename,
891                       dCSRmat *A);
892
893  void fasp_dstr_write (const char *filename,
894                       dSTRmat *A);
895
896  void fasp_dbsr_write (const char *filename,
897                       dBSRmat *A);
898
899  void fasp_dvec_write (const char *filename,
900                        dvector *vec);
901
902  void fasp_dvecind_write (const char *filename,
903                           dvector *vec);
904
905  void fasp_ivec_write (const char *filename,
906                        ivector *vec);
907
908  void fasp_dvec_print (INT n,
909                        dvector *u);
910
911  void fasp_ivec_print (INT n,
912                        ivector *u);
913
914  void fasp_dcsr_print (dCSRmat *A);
915
916  void fasp_dcoo_print (dCOOmat *A);
917
918  void fasp_dbsr_print (dBSRmat *A);
919
920  void fasp_dbsr_write_coo (const char *filename,
921                            const dBSRmat *A);
922
923  void fasp_dcsr_write_coo (const char *filename,
924                            const dCSRmat *A);
925
926  void fasp_dstr_print (dSTRmat *A);
927
928  void fasp_matrix_read (const char *filename,
929                        void *A);
930
931  void fasp_matrix_read_bin (const char *filename,
932                            void *A);
933
934  void fasp_matrix_write (const char *filename,
935                         void *A,
936                         INT  flag);
937
938  void fasp_vector_read (const char *filerhs,
939                        void *b);
```

```
940
941  void fasp_vector_write (const char *filerhs,
942                          void *b,
943                          INT  flag);
944
945  void fasp_hb_read (const char *input_file,
946                     dCSRmat *A,
947                     dvector *b);
```

NOTE: The above function decorations are taken from "base/include/fasp_functs.h". This header file is automatically generated based on the source codes. Users are discouraged from changing it by hand; their changes may be lost.

3.4 Sparse BLAS

The matrix-vector multiplication: $y = Ax$ can be performed in the following simple way:

```
1   /**
2    * \fn void fasp_blas_dcsr_mxv (dCSRmat *A, REAL *x, REAL *y)
3    *
4    * \brief Matrix-vector multiplication y = A*x
5    *
6    * \param A   Pointer to dCSRmat matrix A
7    * \param x   Pointer to array x
8    * \param y   Pointer to array y
9    *
10   * \author Chensong Zhang
11   * \date   07/01/2009
12   */
13  void fasp_blas_dcsr_mxv (dCSRmat *A,
14                           REAL *x,
15                           REAL *y)
16  {
17      const INT   m  = A->row;
18      const INT  *ia = A->IA, *ja = A->JA;
19      const REAL *aj = A->val;
20
21      INT i, k, beg, end;
22      register REAL tmp;
23
24      for ( i=0; i<m; ++i ) {
25          tmp = 0.0;
26          beg = ia[i]; end = ia[i+1];
27          for ( k=beg; k<end; ++k ) tmp += aj[k]*x[ja[k]];
28          y[i] = tmp;
29      }
30  }
```

This is only a simple example for sparse matrix-vector multiplication (SpMV) kernel. Since we need many types of sparse matrices, there are various of versions of SpMV for different data structures. See the Reference Manual for more details.

3.5   Iterative methods

In FASP, there are a couple of standard preconditioned iterative methods [15] implemented, including preconditioned CG, BiCGstab, GMRES, Variable Restarting GMRES, Flexible GMRES, etc. In this section, we use the CSR matrix format as example to introduce how to call these iterative methods. To learn more details, we refer to the Reference Manual.

We first notice the abstract interface for the iterative methods. The following code segment is taken from "base/src/itsolver_csr.c":

```
20   /**
21    * \fn INT fasp_solver_dcsr_itsolver (dCSRmat *A, dvector *b, dvector *x,
22    *                                    precond *pc, itsolver_param *itparam)
23    *
24    * \brief Solve Ax=b by preconditioned Krylov methods for CSR matrices
25    *
26    * \param A        Pointer to the coeff matrix in dCSRmat format
27    * \param b        Pointer to the right hand side in dvector format
28    * \param x        Pointer to the approx solution in dvector format
29    * \param pc       Pointer to the preconditioning action
30    * \param itparam  Pointer to parameters for iterative solvers
31    *
32    * \return         Iteration number if converges; ERROR otherwise.
33    *
34    * \author Chensong Zhang
35    * \date    09/25/2009
36    *
37    * \note This is an abstract interface for iterative methods.
38    */
39   INT fasp_solver_dcsr_itsolver (dCSRmat *A,
40                                  dvector *b,
41                                  dvector *x,
42                                  precond *pc,
43                                  itsolver_param *itparam)
```

The names of the input arguments explain themselves mostly and they are explained in the Reference Manual in detail.

We briefly discuss how to call this function; and, once you understand PCG, you can easily call other iterative methods.

```
463       // ILU setup for whole matrix
464       ILU_data LU;
465       if ( (status = fasp_ilu_dcsr_setup(A,&LU,iluparam)) < 0 ) goto FINISHED;
```

```
466
467        // check iludata
468        if ( (status = fasp_mem_iludata_check(&LU)) < 0 ) goto FINISHED;
469
470        // set preconditioner
471        precond pc;
472        pc.data = &LU;
473        pc.fct  = fasp_precond_ilu;
474
475        // call iterative solver
476        status = fasp_solver_dcsr_itsolver(A,b,x,&pc,itparam);
```

Now we explain this code segment a little bit:

- Line 463–465 performs the setup phase for ILU method. The particular type of ILU method is determined by "iluparam"; see §2.4. Line 7 performs a simple memory check for ILU.

- Line 471–473 defines the preconditioner data structure "pc", which contains two parts: one is the actual preconditioning action "pc.fct", the other is the auxiliary data which is needed to perform the preconditioning action "pc.data".

- Line 476 calls iterative methods. "A" is the matrix in dCSRmat format; "b" and "x" are the right-hand side and the solution vectors, respectively. Similar to ILU setup, the type of iterative methods is determined by "itparam".

Apparently, we now left with no choice but introducing "itparam".

```
/**
 * \struct itsolver_param
 * \brief Parameters passed to iterative solvers
 */
typedef struct {

    SHORT itsolver_type; /**< solver type: see message.h */
    SHORT precond_type;  /**< preconditioner type: see message.h */
    SHORT stop_type;     /**< stopping criteria type */
    INT   maxit;         /**< max number of iterations */
    REAL  tol;           /**< convergence tolerance */
    INT   restart;       /**< number of steps for restarting: for GMRES etc */
    SHORT print_level;   /**< print level: 0--10 */

} itsolver_param; /**< Parameters for iterative solvers */
```

Possible "itsolver_type" includes:

```
/**
 * \brief Definition of solver types for iterative methods
 */
#define SOLVER_DEFAULT          0  /**< Use default solver in FASP */
```

```
//-------------------------------------------------------------------------------
#define SOLVER_CG                  1   /**< Conjugate Gradient */
#define SOLVER_BiCGstab            2   /**< Bi-Conjugate Gradient Stabilized */
#define SOLVER_MinRes              3   /**< Minimal Residual */
#define SOLVER_GMRES               4   /**< Generalized Minimal Residual */
#define SOLVER_VGMRES              5   /**< Variable Restarting GMRES */
#define SOLVER_VFGMRES             6   /**< Variable Restarting Flexible GMRES */
#define SOLVER_GCG                 7   /**< Generalized Conjugate Gradient */
#define SOLVER_GCR                 8   /**< Generalized Conjugate Residual */
//-------------------------------------------------------------------------------
#define SOLVER_SCG                11   /**< Conjugate Gradient with safe net */
#define SOLVER_SBiCGstab          12   /**< BiCGstab with safe net */
#define SOLVER_SMinRes            13   /**< MinRes with safe net */
#define SOLVER_SGMRES             14   /**< GMRes with safe net */
#define SOLVER_SVGMRES            15   /**< Variable-restart GMRES with safe net */
#define SOLVER_SVFGMRES           16   /**< Variable-restart FGMRES with safe net */
#define SOLVER_SGCG               17   /**< GCG with safe net */
//-------------------------------------------------------------------------------
#define SOLVER_AMG                21   /**< AMG as an iterative solver */
#define SOLVER_FMG                22   /**< Full AMG as an solver */
```

3.6   Geometric multigrid

The geometric multigrid method (GMG) is one of the most efficient solving techniques for discrete algebraic systems arising from many types of partial differential equations [1, 16]. GMG utilizes a hierarchy of grids or discretizations and reduces the error at a number of frequencies simultaneously. Because of its plausible linear complexity—i.e., the low computational cost of solving a linear system with N unknowns is $O(N)$—the GMG method is one of the most popular Poisson solvers. Although the GMG's applicability is limited as it requires explicit information on the hierarchy of the discrete system, when it can be applied, GMG is far more efficient than its algebraic version, the algebraic multigrid (AMG) method.

We now give a simple example on calling the geometric multigrid for solving the Poisson's equation in 2D (discretized by the standard five-point finite difference stencil). Consider the Poisson equation

$$
\begin{cases}
-\Delta u & = & f & \text{in } \Omega \\
u & = & 0 & \text{on } \partial\Omega,
\end{cases}
$$

where $\Omega = (0,1)^2 \subset \mathbb{R}^2$. The main reason why we choose this simplest possible setting is to emphasize that, even for a simple problem, the new heterogeneous architectures present challenges for numerical implementation. Another reason is to allow us to use explicit stencils and to avoid the bottleneck of sparse matrix-vector production. The standard central finite difference method is applied to discretize the Poisson's equation. In other words, the Laplace operator is discretized by the classical second-order central difference scheme. After discretization, we end up with a system

of linear equations:

$$\mathbf{A}\vec{u} = \vec{f}.$$

We use the five-point central difference scheme in 2D. Consider a uniform square mesh of $\Omega = [0,1]^2$ with size $h = \frac{1}{n}$ and in which $x_i = ih$, $y_j = jh$ $(i, j = 0, 1, \ldots, n)$. Let $u_{i,j}$ be the numerical approximation of $u(x_i, y_j)$. The five-point central difference scheme for solving the Poisson's equation in 2D can be written as follows:

$$-u_{i-1,j} - u_{i,j-1} + 4u_{i,j} - u_{i+1,j} - u_{i,j+1} = h^2 f(x_i, y_j) \qquad i, j = 1, 2, \ldots, n - 1.$$

The sample code for this solver can be found in "`test/main/testgmg.c`" and a piece of the source code is listed as follows:

```
254          case 2: // 2 dimension
255
256              u = (REAL *)malloc((nx+1)*(ny+1)*sizeof(REAL));
257              fasp_array_set((nx+1)*(ny+1), u, 0.0);
258
259              b = (REAL *)malloc((nx+1)*(ny+1)*sizeof(REAL));
260              for (i = 0; i <= nx; i++) {
261                  for (j = 0; j <= ny; j++) {
262                      b[j*(nx+1)+i] = h*h*f2d(i, j, nx, ny);
263                  }
264              }
265
266              switch (method) {
267
268                  case 1: // V-cycle
269                      fasp_poisson_gmg_2D(u, b, nx, ny, maxlevel, rtol, prtlvl);
270                      break;
271
272                  case 2: // FMG
273                      fasp_poisson_fgmg_2D(u, b, nx, ny, maxlevel, rtol, prtlvl);
274                      break;
275
276                  case 3: // PCG
277                      fasp_poisson_pcg_gmg_2D(u, b, nx, ny, maxlevel, rtol, prtlvl);
278                      break;
279
280              }
281
282          break;
```

3.7   Algebraic multigrid

The classical algebraic multigrid method [14] is an important component in many of our auxiliary space preconditioners. Because of its user-friendly and scalability, AMG becomes increasingly

popular in scientific and engineering computing, especially when GMG is difficult or not possible to be applied. Various of new AMG techniques [17, 18, 4, 7, 5, 9, 6, 20, 3, 11, 8] have emerged in recent years.

The following code segment is part of "base/src/amg.c" and it is a good example which shows how to call different AMG methods (classical AMG, smoothed aggregation, un-smoothed aggregation) and different multilevel iterative methods (V-cycle, W-cycle, AMLI-cycle, Nonlinear AMLI-cycle, etc).

```
42      const SHORT    max_levels  = param->max_levels;
43      const SHORT    prtlvl      = param->print_level;
44      const SHORT    amg_type    = param->AMG_type;
45      const SHORT    cycle_type  = param->cycle_type;
46      const INT      nnz = A->nnz, m = A->row, n = A->col;
47
48      // local variables
49      SHORT          status;
50      AMG_data *     mgl = fasp_amg_data_create(max_levels);
51      REAL           AMG_start, AMG_end;
52
53  #if DEBUG_MODE > 0
54      printf("### DEBUG: %s ...... [Start]\n", __FUNCTION__);
55  #endif
56
57      if ( prtlvl > PRINT_NONE ) fasp_gettime(&AMG_start);
58
59      // check matrix data
60      if ( m != n ) {
61          printf("### ERROR: A is not a square matrix!\n");
62          fasp_chkerr(ERROR_MAT_SIZE, __FUNCTION__);
63      }
64
65      if ( nnz <= 0 ) {
66          printf("### ERROR: A has no nonzero entries!\n");
67          fasp_chkerr(ERROR_MAT_SIZE, __FUNCTION__);
68      }
69
70      // Step 0: initialize mgl[0] with A, b and x
71      mgl[0].A = fasp_dcsr_create(m, n, nnz);
72      fasp_dcsr_cp(A, &mgl[0].A);
73
74      mgl[0].b = fasp_dvec_create(n);
75      fasp_dvec_cp(b, &mgl[0].b);
76
77      mgl[0].x = fasp_dvec_create(n);
78      fasp_dvec_cp(x, &mgl[0].x);
79
80      // Step 1: AMG setup phase
81      switch (amg_type) {
82
83          case SA_AMG: // Smoothed Aggregation AMG setup
```

```
84              if ( prtlvl > PRINT_NONE ) printf("\nCalling SA AMG ...\n");
85              status = fasp_amg_setup_sa(mgl, param); break;
86
87          case UA_AMG: // Unsmoothed Aggregation AMG setup
88              if ( prtlvl > PRINT_NONE ) printf("\nCalling UA AMG ...\n");
89              status = fasp_amg_setup_ua(mgl, param); break;
90
91          default: // Classical AMG setup
92              if ( prtlvl > PRINT_NONE ) printf("\nCalling classical AMG ...\n");
93              status = fasp_amg_setup_rs(mgl, param);
94
95      }
96
97      // Step 2: AMG solve phase
98      if ( status == FASP_SUCCESS ) { // call a multilevel cycle
99
100         switch (cycle_type) {
101
102             case AMLI_CYCLE: // AMLI-cycle
103                 fasp_amg_solve_amli(mgl, param); break;
104
105             case NL_AMLI_CYCLE: // Nonlinear AMLI-cycle
106                 fasp_amg_solve_nl_amli(mgl, param); break;
107
108             default: // V,W-cycles (determined by param)
109                 fasp_amg_solve(mgl, param); break;
110
111         }
112
113         fasp_dvec_cp(&mgl[0].x, x);
114
115     }
116
117     else { // call a backup solver
118
119         if ( prtlvl > PRINT_MIN ) {
120             printf("### WARNING: AMG setup failed!\n");
121             printf("### WARNING: Use a backup solver instead.\n");
122         }
123         fasp_solver_dcsr_spgmres (A, b, x, NULL, param->tol, param->maxit,
124                                   20, 1, prtlvl);
125
126     }
```

The code above is very simple and we only wish to point out that:

- Line 42–45 reads some of the parameters from "AMG_param", which can be defined from a input file; see §2.4.

- Line 50–78 initializes the "AMG_data" with a copy of the coefficient matrix, the right-hand side, and the initial solution (it will store the final solution eventually).

- Line 81–95 calls three different AMG setup methods, determined by "amg_type".

- Line 98–115 calls three different multilevel iterative methods, determined by "cycle_type".

Parameters for AMG

There are a couple of controlling parameters for algebraic multigrid methods in FASP. Basically, there are four types of parameters for AMG—They control multilevel iterations, smoothing, classical AMG setup, and aggregation AMG setup. The following is a sample from "test/ini/input.dat" and a brief explanation of each parameter is given.

```
55  %——————————————————————————————%
56  % parameters for multilevel iteration             %
57  %——————————————————————————————%
58
59  AMG_type                  = C       % C classic AMG
60                                      % SA smoothed aggregation
61                                      % UA unsmoothed aggregation
62  AMG_cycle_type            = V       % V V-cycle | W W-cycle
63                                      % A AMLI-cycle | NA Nonlinear AMLI-cycleA
64  AMG_tol                   = 1e-6    % tolerance for AMG
65  AMG_maxit                 = 1       % number of AMG iterations
66  AMG_levels                = 20      % max number of levels
67  AMG_coarse_dof            = 500     % max number of coarse degrees of freedom
68  AMG_coarse_solver         = 0       % coarsest solver: 0 iterative |
69                                      % 31 SuperLU | 32 UMFPack | 33 MUMPS
70  AMG_coarse_scaling        = OFF     % switch of scaling of the coarse grid correction
71  AMG_amli_degree           = 2       % degree of the polynomial used by AMLI cycle
72  AMG_nl_amli_krylov_type   = 6       % Krylov method in NLAMLI cycle: 6 FGMRES | 7 GCG
73
74  %——————————————————————————————%
75  % parameters for AMG smoothing                    %
76  %——————————————————————————————%
77
78  AMG_smoother              = GS      % GS | JACOBI | SGS SOR | SSOR |
79                                      % GSOR | SGSOR | POLY | L1DIAG | CG
80  AMG_smooth_order          = CF      % NO: natural order | CF: CF order
81  AMG_ILU_levels            = 0       % number of levels using ILU smoother
82  AMG_Schwarz_levels        = 0       % number of levels using Schwarz smoother
83  AMG_relaxation            = 1.0     % relaxation parameter for SOR smoother
84  AMG_polynomial_degree     = 3       % degree of the polynomial smoother
85  AMG_presmooth_iter        = 1       % number of presmoothing sweeps
86  AMG_postsmooth_iter       = 1       % number of postsmoothing sweeps
87
88  %——————————————————————————————%
89  % parameters for classical AMG SETUP              %
90  %——————————————————————————————%
91
92  AMG_coarsening_type       = 1       % 1 Modified RS
93                                      % 3 Compatible Relaxation
94                                      % 4 Aggressive
```

```
 95   AMG_interpolation_type   = 1       % 1 Direct | 2 Standard | 3 Energy−min
 96   AMG_strong_threshold     = 0.3     % Strong threshold
 97   AMG_truncation_threshold = 0.1     % Truncation threshold
 98   AMG_max_row_sum          = 0.9     % Max row sum
 99
100   %——————————————————————————————————————————%
101   % parameters for aggregation−type AMG SETUP    %
102   %——————————————————————————————————————————%
103
104   AMG_aggregation_type     = 2       % 1 Matching | 2 VMB
105   AMG_pair_number          = 2       % Number of pairs in matching
106   AMG_strong_coupled       = 0.08    % Strong coupled threshold
107   AMG_max_aggregation      = 20      % Max size of aggregations
108   AMG_tentative_smooth     = 0.67    % Smoothing factor for tentative prolongation
109   AMG_smooth_filter        = OFF     % Switch for filtered matrix for smoothing
110   AMG_quality_bound        = 8.0     % quality of aggregation: 8.0 sysmm | 10.0 unsymm
```

NOTE: Here we can not discuss the details of these parameters as a full discussion requires more understand of the underlying algorithms which we have completely omitted. So to learn more about, we refer to the Reference Manual.

Chapter 4

More Advanced Usage

In this chapter, we discuss a few more advanced features of FASP. We will discuss parallel versions of FASP and its build-in features for debugging purposes. These features will be helpful for people who would like to develop on the top of FASP. For users who only wish to call a few standard solvers, they can skip this chapter.

4.1   An OpenMP example

OpenMP[1] (Open Multiprocessing) is an API that supports multi-platform shared memory multiprocessing programming in C, C++, and Fortran, on most processor architectures and operating systems. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior. Some preliminary OpenMP support has been included since the very beginning of FASP. We consistently improves and expands OpenMP support as multiprocessor architectures become the dominant desktop computing environment.

NOTE: By default, OpenMP is disabled in FASP. In order to turn it on, you need to modify FASP.mk slightly as follows.

To enable OpenMP support, you need to uncomment one line in "FASP.mk" and set "openmp" to be "yes".

```
37  #
38  # You may use multithread version after you enable OpenMP support. To
39  # setup the environment, you need
40  #  >> export OMP_NUM_THREADS=4 (for bash)
41  #  >> setenv OMP_NUM_THREADS 4 (for tcsh)
42  # If you want to compile with OpenMP support, uncomment the next line:
43  #
```

---

[1] Official website: http://openmp.org/

```
44  # openmp=yes
45  #
```

After you build FASP with "openmp=yes", OpenMP is turned on and the number of threads is determined by the environment variable OMP_NUM_THREADS. For example, to use 8 threads in sh/bash, you need to set:

```
$ export OMP_NUM_THREADS=8
```

4.2   Predefined constants

It is important to notice that there are several predefined constants in FASP. Using these macros makes the program more uniform. These constants are defined in "base/include/fasp_const.h":

```
1   /*! \file fasp_const.h
2    *  \brief Definition of all kinds of messages, including error messages,
3    *         solver types, etc.
4    *
5    *  \note  This is internal use only. Do NOT change.
6    *
7    *-------------------------------------------------------------------------------
8    *  Created  by Chensong Zhang on 03/20/2010.
9    *  Modified by Chensong Zhang on 12/06/2011.
10   *  Modified by Chensong Zhang on 12/25/2011.
11   *  Modified by Chensong Zhang on 04/22/2012.
12   *  Modified by Ludmil Zikatanov on 02/15/2013: CG -> SMOOTHER_CG.
13   *  Modified by Chensong Zhang on 02/16/2013: GS -> SMOOTHER_GS, etc.
14   *  Modified by Chensong Zhang on 04/09/2013: Add safe Krylov methods.
15   *  Modified by Chensong Zhang on 09/22/2013: Clean up Doxygen.
16   *  Modified by Chensong Zhang on 09/17/2013: Filename changed from message.h.
17   *-------------------------------------------------------------------------------
18   *
19   */
20
21  #ifndef __FASP_MESSAGES__          /*-- allow multiple inclusions --*/
22  #define __FASP_MESSAGES__
23
24  /**
25   * \brief Definition of return status and error messages
26   */
27  #define FASP_SUCCESS             0  /**< return from function successfully */
28  //--------------------------------------------------------------------------------
29  #define ERROR_OPEN_FILE        -10  /**< fail to open a file */
30  #define ERROR_WRONG_FILE       -11  /**< input contains wrong format */
31  #define ERROR_INPUT_PAR        -13  /**< wrong input argument */
32  #define ERROR_REGRESS          -14  /**< regression test fail */
33  #define ERROR_MAT_SIZE         -15  /**< wrong problem size */
34  #define ERROR_NUM_BLOCKS       -18  /**< wrong number of blocks */
```

```
35  #define ERROR_MISC              -19  /**< other error */
36  //————————————————————————————————————————————————————
37  #define ERROR_ALLOC_MEM         -20  /**< fail to allocate memory */
38  #define ERROR_DATA_STRUCTURE    -21  /**< problem with data structures */
39  #define ERROR_DATA_ZERODIAG     -22  /**< matrix has zero diagonal entries */
40  #define ERROR_DUMMY_VAR         -23  /**< unexpected input data */
41  //————————————————————————————————————————————————————
42  #define ERROR_AMG_INTERP_TYPE -30  /**< unknown interpolation type */
43  #define ERROR_AMG_SMOOTH_TYPE -31  /**< unknown smoother type */
44  #define ERROR_AMG_COARSE_TYPE -32  /**< unknown coarsening type */
45  #define ERROR_AMG_COARSEING   -33  /**< coarsening step failed to complete */
46  //————————————————————————————————————————————————————
47  #define ERROR_SOLVER_TYPE       -40  /**< unknown solver type */
48  #define ERROR_SOLVER_PRECTYPE   -41  /**< unknown precond type */
49  #define ERROR_SOLVER_STAG       -42  /**< solver stagnates */
50  #define ERROR_SOLVER_SOLSTAG    -43  /**< solver's solution is too small */
51  #define ERROR_SOLVER_TOLSMALL   -44  /**< solver's tolerance is too small */
52  #define ERROR_SOLVER_ILUSETUP   -45  /**< ILU setup error */
53  #define ERROR_SOLVER_MISC       -46  /**< misc solver error during run time */
54  #define ERROR_SOLVER_MAXIT      -48  /**< maximal iteration number exceeded */
55  #define ERROR_SOLVER_EXIT       -49  /**< solver does not quit successfully */
56  //————————————————————————————————————————————————————
57  #define ERROR_QUAD_TYPE         -60  /**< unknown quadrature type */
58  #define ERROR_QUAD_DIM          -61  /**< unsupported quadrature dim */
59  //————————————————————————————————————————————————————
60  #define ERROR_LIC_TYPE          -80  /**< wrong license type */
61  //————————————————————————————————————————————————————
62  #define ERROR_UNKNOWN           -99  /**< an unknown error type */
63
64  /**
65   * \brief Definition of logic type
66   */
67  #define TRUE                    1  /**< logic TRUE */
68  #define FALSE                   0  /**< logic FALSE */
69
70  /**
71   * \brief Definition of switch
72   */
73  #define ON                      1  /**< turn on certain parameter */
74  #define OFF                     0  /**< turn off certain parameter */
75
76  /**
77   * \brief Print level for all subroutines —— not including DEBUG output
78   */
79  #define PRINT_NONE              0  /**< silent: no printout at all */
80  #define PRINT_MIN               1  /**< quiet:  print error, important warnings */
81  #define PRINT_SOME              2  /**< some:   print less important warnings */
82  #define PRINT_MORE              4  /**< more:   print some useful debug info */
83  #define PRINT_MOST              8  /**< most:   maximal printouts, no files */
84  #define PRINT_ALL               10 /**< all:    all printouts, including files */
85
86  /**
87   * \brief Definition of matrix format
```

```
 88   **/
 89   #define MAT_FREE              0   /**< matrix-free format: only mxv action */
 90   #define MAT_CSR               1   /**< compressed sparse row */
 91   #define MAT_BSR               2   /**< block-wise compressed sparse row */
 92   #define MAT_STR               3   /**< structured sparse matrix */
 93   #define MAT_bCSR              4   /**< block matrix of CSR */
 94   #define MAT_bBSR              5   /**< block matrix of BSR for bordered systems */
 95   #define MAT_CSRL              6   /**< modified CSR to reduce cache missing */
 96   #define MAT_SymCSR            7   /**< symmetric CSR format */
 97
 98   /**
 99    * \brief Definition of solver types for iterative methods
100    */
101   #define SOLVER_DEFAULT        0   /**< Use default solver in FASP */
102   //—————————————————————————————————————————————————————————————————————
103   #define SOLVER_CG             1   /**< Conjugate Gradient */
104   #define SOLVER_BiCGstab       2   /**< Bi-Conjugate Gradient Stabilized */
105   #define SOLVER_MinRes         3   /**< Minimal Residual */
106   #define SOLVER_GMRES          4   /**< Generalized Minimal Residual */
107   #define SOLVER_VGMRES         5   /**< Variable Restarting GMRES */
108   #define SOLVER_VFGMRES        6   /**< Variable Restarting Flexible GMRES */
109   #define SOLVER_GCG            7   /**< Generalized Conjugate Gradient */
110   #define SOLVER_GCR            8   /**< Generalized Conjugate Residual */
111   //—————————————————————————————————————————————————————————————————————
112   #define SOLVER_SCG            11  /**< Conjugate Gradient with safe net */
113   #define SOLVER_SBiCGstab      12  /**< BiCGstab with safe net */
114   #define SOLVER_SMinRes        13  /**< MinRes with safe net */
115   #define SOLVER_SGMRES         14  /**< GMRes with safe net */
116   #define SOLVER_SVGMRES        15  /**< Variable-restart GMRES with safe net */
117   #define SOLVER_SVFGMRES       16  /**< Variable-restart FGMRES with safe net */
118   #define SOLVER_SGCG           17  /**< GCG with safe net */
119   //—————————————————————————————————————————————————————————————————————
120   #define SOLVER_AMG            21  /**< AMG as an iterative solver */
121   #define SOLVER_FMG            22  /**< Full AMG as an solver */
122   //—————————————————————————————————————————————————————————————————————
123   #define SOLVER_SUPERLU        31  /**< SuperLU Direct Solver */
124   #define SOLVER_UMFPACK        32  /**< UMFPack Direct Solver */
125   #define SOLVER_MUMPS          33  /**< MUMPS   Direct Solver */
126
127   /**
128    * \brief Definition of iterative solver stopping criteria types
129    */
130   #define STOP_REL_RES          1   /**< relative residual ||r||/||b|| */
131   #define STOP_REL_PRECRES      2   /**< relative B-residual ||r||_B/||b||_B */
132   #define STOP_MOD_REL_RES      3   /**< modified relative residual ||r||/||x|| */
133
134   /**
135    * \brief Definition of preconditioner type for iterative methods
136    */
137   #define PREC_NULL             0   /**< with no precond */
138   #define PREC_DIAG             1   /**< with diagonal precond */
139   #define PREC_AMG              2   /**< with AMG precond */
140   #define PREC_FMG              3   /**< with full AMG precond */
```

```
141   #define PREC_ILU              4   /**< with ILU precond */
142   #define PREC_SCHWARZ          5   /**< with Schwarz preconditioner */
143
144   /**
145    * \brief Type of ILU methods
146    */
147   #define ILUk                  1   /**< ILUk */
148   #define ILUt                  2   /**< ILUt */
149   #define ILUtp                 3   /**< ILUtp */
150
151   /**
152    * \brief Type of Schwarz smoother
153    */
154   #define SCHWARZ_FORWARD       1   /**< Forward ordering */
155   #define SCHWARZ_BACKWARD      2   /**< Backward ordering */
156   #define SCHWARZ_SYMMETRIC     3   /**< Symmetric smoother */
157
158   /**
159    * \brief Definition of AMG types
160    */
161   #define CLASSIC_AMG           1   /**< classic AMG */
162   #define SA_AMG                2   /**< smoothed aggregation AMG */
163   #define UA_AMG                3   /**< unsmoothed aggregation AMG */
164
165   /**
166    * \brief Definition of aggregation types
167    */
168   #define PAIRWISE              1   /**< pairwise aggregation */
169   #define VMB                   2   /**< VMB aggregation */
170
171   /**
172    * \brief Definition of cycle types
173    */
174   #define V_CYCLE               1   /**< V-cycle */
175   #define W_CYCLE               2   /**< W-cycle */
176   #define AMLI_CYCLE            3   /**< AMLI-cycle */
177   #define NL_AMLI_CYCLE         4   /**< Nonlinear AMLI-cycle */
178
179   /**
180    * \brief Definition of standard smoother types
181    */
182   #define SMOOTHER_JACOBI       1   /**< Jacobi smoother */
183   #define SMOOTHER_GS           2   /**< Gauss-Seidel smoother */
184   #define SMOOTHER_SGS          3   /**< Symmetric Gauss-Seidel smoother */
185   #define SMOOTHER_CG           4   /**< CG as a smoother */
186   #define SMOOTHER_SOR          5   /**< SOR smoother */
187   #define SMOOTHER_SSOR         6   /**< SSOR smoother */
188   #define SMOOTHER_GSOR         7   /**< GS + SOR smoother */
189   #define SMOOTHER_SGSOR        8   /**< SGS + SSOR smoother */
190   #define SMOOTHER_POLY         9   /**< Polynomial smoother */
191   #define SMOOTHER_L1DIAG       10  /**< L1 norm diagonal scaling smoother */
192
193   /**
```

```
194    * \brief Definition of coarsening types
195    */
196   #define COARSE_RS            1   /**< Classical */
197   #define COARSE_RSP           2   /**< Classical, with positive offdiags */
198   #define COARSE_CR            3   /**< Compatible relaxation */
199   #define COARSE_AC            4   /**< Aggressive coarsening */
200   #define COARSE_MIS           5   /**< Aggressive coarsening based on MIS */
201
202   /**
203    * \brief Definition of interpolation types
204    */
205   #define INTERP_DIR           1   /**< Direct interpolation */
206   #define INTERP_STD           2   /**< Standard interpolation */
207   #define INTERP_ENG           3   /**< energy minimization interpolation */
208
209   /**
210    * \brief Type of vertices (DOFs) for coarsening
211    */
212   #define GOPT                -5   /**< Cannot fit in aggregates */
213   #define UNPT                -1   /**< Undetermined points */
214   #define FGPT                 0   /**< Fine grid points  */
215   #define CGPT                 1   /**< Coarse grid points */
216   #define ISPT                 2   /**< Isolated points */
217
218   /**
219    * \brief Definition of smoothing order
220    */
221   #define NO_ORDER             0   /**< Natural order smoothing */
222   #define CF_ORDER             1   /**< C/F order smoothing */
223
224   /**
225    * \brief Type of ordering for smoothers
226    */
227   #define USERDEFINED          0   /**< User defined order */
228   #define CPFIRST              1   /**< C-points first order */
229   #define FPFIRST             -1   /**< F-points first order */
230   #define ASCEND              12   /**< Ascending order */
231   #define DESCEND             21   /**< Descending order */
232
233   /**
234    * \brief Some global constants
235    */
236   #define BIGREAL           1e+20   /**< A large real number */
237   #define SMALLREAL         1e-20   /**< A small real number */
238   #define SMALLREAL2        1e-40   /**< An extremely small real number */
239   #define MAX_REFINE_LVL       20   /**< Maximal refinement level */
240   #define MAX_AMG_LVL          20   /**< Maximal AMG coarsening level */
241   #define MIN_CDOF             20   /**< Minimal number of coarsest variables */
242   #define MIN_CRATE           0.9   /**< Minimal coarsening ratio */
243   #define MAX_CRATE          20.0   /**< Maximal coarsening ratio */
244   #define MAX_RESTART          20   /**< Maximal restarting number */
245   #define MAX_STAG             20   /**< Maximal number of stagnation times */
246   #define STAG_RATIO         1e-4   /**< Stagnation tolerance = tol*STAGRATIO */
```

```
247  #define OPENMP_HOLDS          2000   /**< Smallest size for OpenMP version */
248
249  #endif                              /* end if for __FASP_MESSAGES__ */
250
251  /*————————————————————————————*/
252  /*——      End of File        ——*/
253  /*————————————————————————————*/
```

4.3   The debug environment

NOTE: By default, a RELEASE version of FASP is compiled and no warnings are displaced during building in FASP.

There is a built-in debug feature which is intended to help developers to locate malfunctions in FASP. In order to turn this on, you need to add an option during the config stage by

```
$ make config debug=all
```

When this debug feature is turned on, there will be a lot more information printed when you call FASP. If you just need to compile a DEBUG version instead of RELEASE, you can do use

```
$ make config debug=yes
```

Bibliography

[1] J. Bramble. *Multigrid methods*, volume 294 of *Pitman Research Notes in Mathematics Series*. Longman Scientific & Technical, Harlow, 1993.

[2] A. Brandt, S. McCormick, and J. Ruge. Algebraic multigrid (AMG) for automatic multigrid solution with application to geodetic computations, Report. *Inst. Comp. Studies Colorado State Univ*, 109:110, 1982.

[3] J. Brannick and L. Zikatanov. Algebraic multigrid methods based on compatible relaxation and energy minimization. *Lecture Notes in Computational Science and Engineering*, 55:15, 2007.

[4] M. Brezina, A. Cleary, R. Falgout, V. Henson, J. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic multigrid based on element interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2000.

[5] T. Chartier, R. Falgout, V. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. Vassilevski. Spectral AMGe (ρAMGe). *SIAM J. Sci. Comput.*, 25(1):1–26, 2003.

[6] R. Falgout and P. Vassilevski. On generalizing the algebraic multigrid framework. *SIAM J. Numer. Anal.*, 42(4):1669–1693 (electronic), 2004.

[7] V. Henson and P. Vassilevski. Element-free AMGe: general algorithms for computing interpolation weights in AMG. *SIAM J. Sci. Comput.*, 23(2):629–650, 2001.

[8] X. Hu, P. S. Vassilevski, and J. Xu. Comparative convergence analysis of nonlinear amli-cycle multigrid. *SIAM J. Num. Anal*, 51(2):1349–1369, 2013.

[9] O. E. Livne. Coarsening by compatible relaxation. *Numer. Linear Algebra Appl.*, 11(2-3):205–227, 2004.

[10] J. Mellor-crummey and J. Garvin. Optimizing Sparse Matrix-Vector Product Computations Using Unroll and Jam. *International Journal of High Performance Computing Applications*, 18(2):225—-236, 2004.

[11] A. Muresan and Y. Notay. Analysis of aggregation-based multigrid. *SIAM Journal on Scientific Computing*, 30(2):1082–1103, 2008.

[12] S. Pissanetzky. *Sparse matrix technology*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1984.

[13] J. Ruge and K. Stuben. Efficient solution of finite difference and finite element equations. In D. J. Paddon and H. Holstein, editors, *Multigrid Methods for Integral and Differential Equations*, volume 3, pages 169–212. Clarendon Press, 1985.

[14] J. Ruge and K. Stüben. Algebraic multigrid. *Multigrid methods*, 3:73–130, 1987.

[15] Y. Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[16] U. Trottenberg, C. Oosterlee, and A. Schüller. Multigrid. *With contributions by A. Brandt, P. Oswald and K. Stuben*, 2001.

[17] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[18] W. Wan, T. Chan, and B. Smith. An energy-minimizing interpolation for robust multigrid methods. *SIAM Journal on Scientific Computing*, 21(4):1632–1649, 2000.

[19] J. Xu. Fast Poisson-Based Solvers for Linear and Nonlinear PDEs Jinchao Xu. In *Proceedings of the International Congress of Mathematicians*, pages 2886–2912, 2010.

[20] J. Xu and L. Zikatanov. On an energy minimizing basis for algebraic multigrid methods. *Computing and Visualization in Science*, 7(3):121–127, 2004.