FASP User Guide

FASP Developers

Version 2.7.7

Contents

Chapter 1

Introduction

1.1 General description

The Fast Auxiliary Space Preconditioning (FASP) package provides C source files[1] to build a library of iterative solvers and preconditioners for the solution of large-scale linear systems of equations. The components of the FASP basic 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 include:

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

Source files in the package are organized in various abstract levels as follows:

- Level 0 (Aux*.c): Auxiliary functions (timing, memory, threading, ...)
- Level 1 (Bla*.c): Basic linear algebra subroutines (SpMV, RAP, ILU, SWZ, ...)
- Level 2 (Itr*.c): Iterative methods and smoothers (Jacobi, GS, SOR, Poly, ...)
- Level 3 (Kry*.c): Krylov iterative methods (CG, BiCGstab, MinRes, GMRES, ...)
- Level 4 (Pre*.c): Preconditioners (GMG, AMG, FAMG, ...)
- Level 5 (Sol*.c): User interface for FASP solvers (Solvers, wrappers, ...)
- Level x (Xtr*.c): Interface to external packages (Mumps, Umfpack, ...)

---

[1]The code is in the C99 (ISO/IEC 9899:1999) compatible.

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

1.2   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 also be based on the involvement of the community. While we have our own plans for expanding FASP's capabilities, we also count on the users' input in providing requests for, as well as, contributions to, the expansion of FASP in different application areas. Our team is ready to provide (or help with) the design and the implementation of efficient solvers based on the FASP kernel to best meet the goals and the requirements of our users.

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; $H(\mathrm{curl})/H(\mathrm{div})$ 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-Segalman models, black-oil model, and magnetohydrodynamics (MHD).

Several of these benchmark problems are included as examples in the 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[2] for

---

[2]Available online at http://www.multigrid.org/fasp. It is also available in "faspsolver/doc/doc.zip".

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

There are several ways to download the FASP source files. We recommend users download the most updated version from the FASP page on SourceForge.

Downloading from Multigrid.org

The most updated version of FASP can be downloaded directly from

http://www.multigrid.org/fasp/download/faspsolver.zip

Downloading from GitHub

FASP was hosted on *BitBucket.org*[3] using Mercurial (Hg)[4]. But now the most updated version is hosted on *GitHub*[5]

https://github.com/FaspDevTeam/faspsolver

First, you need to obtain a free copy of FASP kernel functions from our public Git 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"

$ git clone git@github.com:FaspDevTeam/faspsolver.git
```

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

After a long pause[6], 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 GitHub"
$ git pull
```

---

[3]Official website: https://bitbucket.org/

[4]Official website: http://mercurial.selenic.com/

[5]Official website: https://github.com/

[6]In fact, a very long pause. This is because the initial clone with copy all the history data. Depending on the speed of your network, it could take a few minutes.

1.5 Building and installing the FASP library and examples

FASP has been tested using the compilers and built-in libraries of several Linux distributions (`Cent OS`, `Debian`, `Fedora`, `RedHat`, `Ubuntu`) Mac OS X 10.6 and later (Leopard, Snow Leopard, Lion, Mavericks, Yosemite, El Capitan, Sierra, High Sierra, Mojave, Catalina, Big Sur, Monterey), and Windows (XP, Win 7, Win 10) with several compliers, including `gcc`, `g++`, `clang`, `icc`, `VC++`. FASP also easily links to applications written in Fortran and this has been tested with `gfortran`, `g95`, `ifort` Fortran compilers.

FASP on Linux or Mac OS X

To build the FASP library for these operating systems. Open a terminal window, where you can issue commands from the command line and do the following: (1) go to the main FASP directory (we will refer to it as `$(faspsolver)` from now on); (2) create a new directory for Build; (3) enter the Build directory and then execute:

```
$ cmake ..
$ make install
```

These two commands build and install the FASP library/header files. It installs the library in `$(faspsolver)/lib` and the header files in `$(faspsolver)/include`. It also creates a file `$(faspsolver)/Config.mk` which contains few of the configuration variables and can be loaded by external project Makefiles (see §1.6 for details on `$(faspsolver)Config.mk`).

Note: While these two approaches to build the FASP test suite and the FASP examples produce equivalent result in most cases, we note an important difference. The former approach uses a `CMake` installation process. The latter works without invoking `Cmake` and represents an example of how one may link an external project with the FASP library. We refer to §1.6 below for more details.

If everything went all right, you can go to the "`faspsolver/test`" directory and try to run a test problem:

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

To easily uninstall FASP and clean up the working directory, FASP comes with an automatic uninstallation script; you can run

```
$ make uninstall
$ make clean
```

You can also remove the Build directory after installation.

Many other standard cmake options can be used during the configuration phase. The most frequently used ones include but not limited to the following:

```
$ cmake -DCMAKE_VERBOSE_MAKEFILE=ON ..  // build with verbose on
```

```
$ cmake -DCMAKE_BUILD_TYPE=Debug ..     // build in Debug configuration
```

```
$ cmake CC=clang ..                     // build with specified C compiler
```

```
$ cmake -DUSE_OPENMP=ON ..              // build with OpenMP support
```

```
$ cmake -DUSE_UMFPACK=ON ..             // build with UMFPACK package support
```

```
$ cmake -DUSE_MUMPS=ON ..               // build with MUMPS package support
```

```
$ cmake -DUSE_PARDISO=ON ..             // build with PARDISO package support
```

Windows 7

We provide a Visual Studio 2010 (VS10) distribution and a VS10 distribution of FASP for Windows users. For example, you can just open "faspsolver/vs10/faspsolver-vs10.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 the Visual Studio to build FASP. For example, the build can be accomplished using either Microsoft Visual C++ or Intel C compiler.

If you are using other versions of Visual Studio (like VS05 or VS12), we advise NOT to convert the "VS10" files to your newer VS version automatically because the FASP source files might be removed by the Visual Studio. In such a case, we recommend that you create your own version to build all the libraries and test programs.

If you need to build a VS FASP yourself, you need to 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.
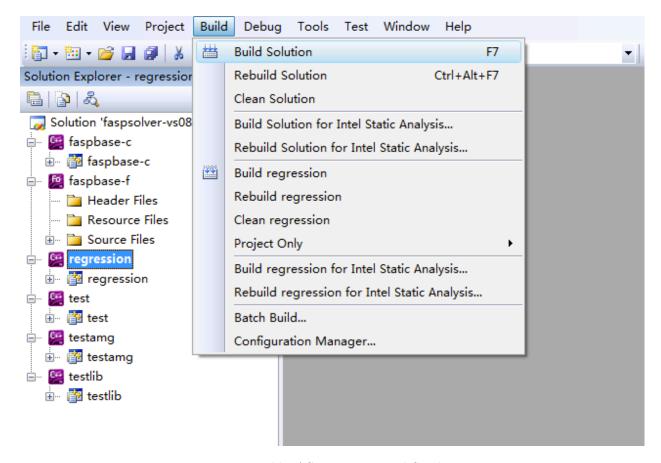
Figure 1.1: Build FASP using Visual Studio 2010.

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 a successful build on VS, you will have two static libraries named "faspbase-c-vs08.lib" and "faspbase-f-vs08.lib". You can use the "lib" command to wrap them 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
```

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 several of them, for example, FASP can be linked to use `UMFPack`, `SuperLU`, `MUMPS`, `SparseKit`, `dlmalloc`.

1.6 Linking your own project with FASP

The FASP distribution comes with two "local" Makefile(s) in the sub-directories for "test" and "tutorial", namely, `$(faspsolver)/test/Makefile` and `$(faspsolver)/tutorial/Makefile`. These two makefiles can be used to build the FASP tests and tutorial examples. They use minimal information about the library built. For convenience, at configuration time, such information is stored in `$(faspsolver)/Config.mk` file for later use. A typical contents of such file is given below:

```
########################### Automatically generated ##################
# Fast Auxiliary Space Preconditioners (FASP)
#
######################################################################
# This file is rewritten when "make config" is run.
# It is (s)included by test/Makefile and tutorial/Makefile
#
######################################################################
fasp_prefix=/path/to/faspsolver
fasp_library=libfasp.a
CC=gcc
CXX=g++
FC=gfortran
######################################################################
```

The variables defined in this file can also be set directly in `$(faspsolver)/test/Makefile`, `$(faspsolver)/tutorial/Makefile`, or, in your own Makefile. An external project can be compiled and linked with FASP by following the rules set in `$(faspsolver)/tutorial/Makefile` (included below):

```
1  ######################################################################
2  # Fast Auxiliary Space Preconditioners (FASP)
3  #
4  ######################################################################
5  # Makefile for building the tutorial executables. If the file
6  # ../Config.mk exists it will take the values of configuration
7  # variables from there. Otherwise, they have to be user supplied below.
8  ######################################################################
```

```
 9
10  fasp_prefix = not-defined-yet
11  fasp_library = not-defined-yet
12  CC = not-defined-yet
13  FC = not-defined-yet
14  CXX = not-defined-yet
15
16  # include the configuration written by CMake at config time if found
17  sinclude ../Config.mk
18
19  ifeq ($(fasp_prefix),not-defined-yet)
20      fasp_prefix = ..
21  endif
22  ifeq ($(fasp_library),not-defined-yet)
23      fasp_library = libfasp.a
24  endif
25  ifeq ($(CC),not-defined-yet)
26     CC=gcc
27  endif
28  ifeq ($(FC),not-defined-yet)
29     FC=gfortran
30  endif
31  ifeq ($(CXX),not-defined-yet)
32     CXX=g++
33  endif
34
35  CFLAGS=-I$(fasp_prefix)/include
36  CFLAGS+=-O3
37  FFLAGS=-I$(fasp_prefix)/include
38  FFLAGS+=-O3
39  LINKER = $(FC) # because of linking with Fortran files
40  LFLAGS = -L$(fasp_prefix)/lib -lfasp
41  fasp_lib_file=$(fasp_prefix)/lib/$(fasp_library)
42
43  examples = poisson-amg-c.ex poisson-its-c.ex poisson-pcg-c.ex \
44              poisson-gmg-c.ex spe01-its-c.ex \
45              poisson-amg-f.ex poisson-pcg-f.ex
46  examples_f = $(filter %-f.ex,$(examples))
47  examples_c = $(filter-out %-f.ex,$(examples))
48
49  .PHONY: all clean
50
51  all:    $(examples_c) $(examples_f)
52
53  %-c.ex: main/%.c $(fasp_lib_file)
54      @$(CC) -c $(CFLAGS) -o main/$@.o $<
55      @$(LINKER) -o $@ main/$@.o $(LFLAGS)
56      @echo 'Building executable file $@'
57
58  %-f.ex: main/%.f90 $(fasp_lib_file)
59      @$(FC) -c $(FFLAGS) -o main/$@.o $<
60      @$(FC) -o $@ main/$@.o $(LFLAGS)
61      @echo 'Building executable file $@'
```

```
62
63  $(fasp_lib_file):
64      $(error The FASP library $@ is not found)
65
66  clean:
67      @-rm -f *.o main/*.o *~
68
69  distclean: clean
70      @-rm -f poisson-amg-c.ex poisson-its-c.ex poisson-pcg-c.ex \
71                 poisson-gmg-c.ex spe01-its-c.ex \
72                 poisson-amg-f.ex poisson-pcg-f.ex
```

Chapter 2

A brief tutorial

In this chapter, we discuss several simple examples included with this FASP distribution and demonstrating how to use the FASP package for solving linear systems. We read the matrices from disk files (the files are also included in the FASP distribution). All the examples in this section can be found inside "`faspsolver/tutorial/`".

After you successfully build FASP (see §1.5), just go to the "`faspsolver/tutorial/`" directory and the tutorial examples should be ready to run.

In this section we mainly discuss the C version of these examples; the FASP distribution also includes F90 versions of some of the examples.

> In the description below, we display a typical output from runs of each of the examples. Note that the actual output depends on the solver parameters, and, on your computer it may be be different than what you see here.

2.1 Example 1: An AMG solver for the Poisson equation

The first example is a standard one: We read a symmetric positive definite matrix A and right-hand side b from harddisk and then we solve $Ax = b$ using the classical AMG method [2, 13, 14]; see §3.6. In this example the matrix A included with the FASP distribution corresponds to a discretization with continuous piecewise linear finite elements of the Poisson equation

$$-\Delta u = f$$

(with the Dirichlet boundary conditions) on a triangulation of a 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.
```

```c
 5    *        C version.
 6    *
 7    *  \note  Solving the Poisson equation (P1 FEM) with AMG: C version
 8    *
 9    *---------------------------------------------------------------------------
10    *  Copyright (C) 2011--Present by the FASP team. All rights reserved.
11    *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
12    *---------------------------------------------------------------------------
13    */
14
15   #include "fasp.h"
16   #include "fasp_functs.h"
17
18   /**
19    * \fn int main (int argc, const char * argv[])
20    *
21    * \brief This is the main function for the first example.
22    *
23    * \author Chensong Zhang
24    * \date   12/21/2011
25    *
26    * Modified by Chensong Zhang on 09/22/2012
27    */
28   int main (int argc, const char * argv[])
29   {
30       input_param      inparam;  // parameters from input files
31       AMG_param        amgparam; // parameters for AMG
32
33       printf("\n========================================");
34       printf("\n||   FASP: AMG example -- C version   ||");
35       printf("\n========================================\n\n");
36
37       // Step 0. Set parameters: We can use ini/amg.dat
38       fasp_param_set(argc, argv, &inparam);
39       fasp_param_init(&inparam, NULL, &amgparam, NULL, NULL);
40
41       // Set local parameters using the input values
42       const int print_level = inparam.print_level;
43
44       // Step 1. Get stiffness matrix and right-hand side
45       // Read A and b -- P1 FE discretization for Poisson. The location
46       // of the data files is given in "ini/amg.dat".
47       dCSRmat A;
48       dvector b, x;
49       char filename1[512], *datafile1;
50       char filename2[512], *datafile2;
51
52       // Read the stiffness matrix from matFE.dat
53       memcpy(filename1, inparam.workdir, STRLEN);
54       datafile1="csrmat_FE.dat"; strcat(filename1, datafile1);
55
56       // Read the RHS from rhsFE.dat
57       memcpy(filename2, inparam.workdir, STRLEN);
```

```
58       datafile2="rhs_FE.dat"; strcat(filename2, datafile2);
59
60       fasp_dcsrvec_read2(filename1, filename2, &A, &b);
61
62       // Step 2. Print problem size and AMG parameters
63       if (print_level>PRINT_NONE) {
64           printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
65           printf("b: n = %d\n", b.row);
66           fasp_param_amg_print(&amgparam);
67       }
68
69       // Step 3. Solve the system with AMG as an iterative solver
70       // Set the initial guess to be zero and then solve it
71       // with AMG method as an iterative procedure
72       fasp_dvec_alloc(A.row, &x);
73       fasp_dvec_set(A.row, &x, 0.0);
74
75       fasp_solver_amg(&A, &b, &x, &amgparam);
76
77       // Step 4. Clean up memory
78       fasp_dcsr_free(&A);
79       fasp_dvec_free(&b);
80       fasp_dvec_free(&x);
81
82       return FASP_SUCCESS;
83   }
84
85   /*--------------------------------*/
86   /*--       End of File        --*/
87   /*--------------------------------*/
```

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

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

- Line 15–16 includes the main FASP header file "fasp.h" and FASP function declarations header "fasp_functs.h". These two headers shall be included in all files that requires FASP subroutines. Please also be noted that the function declarations in "fasp_functs.h" are automatically generated from the source files by an awk script and we do not recommend modifying this file, since your changes may be lost.

- Line 38–39 sets solver parameters using the default parameters or from the command line options; see more discussions in §2.6. 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.

---

[1]Doxygen http://www.doxygen.org is a useful tool for generating documentation from annotated sources. We use it in FASP development.

- Line 47 defines a sparse matrix A in the compressed sparse row (CSR) format. Line 48 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 60 reads the matrix and the right-hand side from two disk files. Line 54–58 defines the filenames of them.

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

- Line 72–73 allocates memory for the solution vector x and set its initial value to be all zero.

- Line 75 solves $Ax = b$ using the AMG method. Type of the AMG method and other parameters have been given in "amgparam" at Line 36; see §3.6.

- Line 78–80 frees up memory allocated for A, b, and x.

To run this example, type:

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

A sample output is listed as follows:

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


fasp_dcsrvec_read2: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec_read2: 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:              50
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
_____


Setting up 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 0.0044 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 0.0019 seconds.
AMG totally costs 0.0066 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 solver from a Fortran based application can see how to do this example.

```fortran
 1  !> \file  poisson-amg.f90
 2  !>
 3  !> \brief The first test example for FASP: using AMG to solve
 4  !>        the discrete Poisson equation from P1 finite element.
 5  !>        F90 version.
 6  !>
 7  !> \note  Solving the Poisson equation (P1 FEM) with AMG: F90 version
 8  !>
 9  !>------------------------------------------------------------------------------
10  !>  Copyright (C) 2011--Present by the FASP team. All rights reserved.
11  !>  Released under the terms of the GNU Lesser General Public License 3.0 or later.
12  !>------------------------------------------------------------------------------
13
14  program test
15
16    implicit none
```

```fortran
17
18     double precision, dimension(:), allocatable :: u, b, a
19     integer,          dimension(:), allocatable :: ia, ja
20
21     integer          :: iufile, n, nnz, i, prt_lvl, maxit
22     double precision :: tol
23
24     print*, ""
25     write(*,"(A)") "=========================================="
26     write(*,"(A)") "||    FASP: AMG example -- F90 version    ||"
27     write(*,"(A)") "=========================================="
28     print*, ""
29
30     ! Step 0: user defined variables
31     prt_lvl = 3
32     maxit = 100
33     tol = 1.0d-6
34     iufile = 1
35
36     ! Step 1: read A and b
37
38     !===> Read data A from file
39     open(unit=iufile,file='../data/csrmat_FE.dat')
40
41     read(iufile,*) n
42     allocate(ia(1:n+1))
43     read(iufile,*) (ia(i),i=1,n+1)
44
45     nnz=ia(n+1)-ia(1)
46     allocate(ja(1:nnz),a(1:nnz))
47     read(iufile,*) (ja(i),i=1,nnz)
48     read(iufile,*) (a(i),i=1,nnz)
49
50     close(iufile)
51
52     !===> Read data b from file
53     open(unit=iufile,file='../data/rhs_FE.dat')
54
55     read(iufile,*) n
56     allocate(b(1:n))
57     read(iufile,*) (b(i),i=1,n)
58
59     close(iufile)
60
61     !===> Shift the index to start from 0 (for C routines)
62     forall (i=1:n+1) ia(i)=ia(i)-1
63     forall (i=1:nnz) ja(i)=ja(i)-1
64
65     ! Step 2: Solve the system
66
67     !===> Initial guess
68     allocate(u(1:n))
69     u=0.0d0
```

```fortran
70      call fasp_fwrapper_dcsr_amg(n,nnz,ia,ja,a,b,u,tol,maxit,prt_lvl)
71
72      ! Step 3: Clean up memory
73      deallocate(ia,ja,a)
74      deallocate(b,u)
75
76  end program test
77
78  !/*---------------------------------*/
79  !/*--        End of File          --*/
80  !/*---------------------------------*/
```

2.2   Example 2: Conjugate gradient without preconditioning

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
 1  /*! \file  poisson-its.c
 2   *
 3   *  \brief The second test example for FASP: using ITS to solve
 4   *         the discrete Poisson equation from P1 finite element.
 5   *
 6   *  \note  Solving the Poisson equation (P1 FEM) with iterative methods: C version
 7   *
 8   *---------------------------------------------------------------------------------
 9   *  Copyright (C) 2012--Present by the FASP team. All rights reserved.
10   *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
11   *---------------------------------------------------------------------------------
12   */
13
14  #include "fasp.h"
15  #include "fasp_functs.h"
16
17  /**
18   * \fn int main (int argc, const char * argv[])
19   *
20   * \brief This is the main function for the second example.
21   *
22   * \author Feiteng Huang
23   * \date   04/13/2012
24   *
25   * Modified by Chensong Zhang on 09/22/2012
26   */
27  int main (int argc, const char * argv[])
28  {
29      input_param     inparam;  // parameters from input files
30      ITS_param       itparam;  // parameters for itsolver
31
32      printf("\n=======================================");
33      printf("\n||   FASP: ITS example -- C version   ||");
```

```c
34      printf("\n=========================================\n\n");
35
36      // Step 0. Set parameters: We can use ini/its.dat
37      fasp_param_set(argc, argv, &inparam);
38      fasp_param_init(&inparam, &itparam, NULL, NULL, NULL);
39
40      // Set local parameters
41      const int print_level = inparam.print_level;
42
43      // Step 1. Get stiffness matrix and right-hand side
44      // Read A and b -- P1 FE discretization for Poisson. The location
45      // of the data files is given in "ini/its.dat".
46      dCSRmat A;
47      dvector b, x;
48      char filename1[512], *datafile1;
49      char filename2[512], *datafile2;
50
51      // Read the stiffness matrix from matFE.dat
52      memcpy(filename1, inparam.workdir, STRLEN);
53      datafile1="csrmat_FE.dat"; strcat(filename1, datafile1);
54
55      // Read the RHS from rhsFE.dat
56      memcpy(filename2, inparam.workdir, STRLEN);
57      datafile2="rhs_FE.dat"; strcat(filename2, datafile2);
58
59      fasp_dcsrvec_read2(filename1, filename2, &A, &b);
60
61      // Step 2. Print problem size and ITS parameters
62      if (print_level>PRINT_NONE) {
63          printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
64          printf("b: n = %d\n", b.row);
65          fasp_param_solver_print(&itparam);
66      }
67
68      // Step 3. Solve the system with ITS as an iterative solver
69      // Set the initial guess to be zero and then solve it using standard
70      // iterative methods, without applying any preconditioners
71      fasp_dvec_alloc(A.row, &x);
72      fasp_dvec_set(A.row, &x, 0.0);
73
74      fasp_solver_dcsr_itsolver(&A, &b, &x, NULL, &itparam);
75
76      // Step 4. Clean up memory
77      fasp_dcsr_free(&A);
78      fasp_dvec_free(&b);
79      fasp_dvec_free(&x);
80
81      return FASP_SUCCESS;
82  }
83
84  /*---------------------------------*/
85  /*--        End of File          --*/
86  /*---------------------------------*/
```

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

- Line 71–72 allocates memory for the solution vector x and set its initial value to be all zero.

- Line 74 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 as follows:

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


fasp_dcsrvec_read2: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec_read2: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969

        Parameters in ITS_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 CG 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
```

2.3   Example 3: Conjugate gradient with preconditioning

This example is a bit more involved and is a modification of the previous one. In this example, we wish to demonstrate how to use a the FASP library and run a preconditioned conjugate conjugate gradient (PCG) method.

```c
1  /*! \file   poisson-pcg.c
2   *
3   *  \brief The third test example for FASP: using PCG to solve
4   *          the discrete Poisson equation from P1 finite element.
5   *          C version.
6   *
```

```
 7   *   \note   Solving the Poisson equation (P1 FEM) with PCG: C version
 8   *
 9   *---------------------------------------------------------------------------
10   *   Copyright (C) 2012--Present by the FASP team. All rights reserved.
11   *   Released under the terms of the GNU Lesser General Public License 3.0 or later.
12   *---------------------------------------------------------------------------
13   */
14
15  #include "fasp.h"
16  #include "fasp_functs.h"
17
18  /**
19   * \fn int main (int argc, const char * argv[])
20   *
21   * \brief This is the main function for the third example.
22   *
23   * \author Feiteng Huang
24   * \date   05/17/2012
25   *
26   * Modified by Chensong Zhang on 09/22/2012
27   * Modified by Chensong Zhang on 12/23/2018: Fix memory leakage
28   */
29  int main (int argc, const char * argv[])
30  {
31      input_param     inparam;  // parameters from input files
32      ITS_param       itparam;  // parameters for itsolver
33      AMG_param       amgparam; // parameters for AMG
34      ILU_param       iluparam; // parameters for ILU
35
36      printf("\n=======================================");
37      printf("\n||   FASP: PCG example -- C version    ||");
38      printf("\n=======================================\n\n");
39
40      // Step 0. Set parameters: We can use ini/pcg.dat
41      fasp_param_set(argc, argv, &inparam);
42      fasp_param_init(&inparam, &itparam, &amgparam, &iluparam, NULL);
43
44      // Set local parameters
45      const SHORT print_level = itparam.print_level;
46      const SHORT pc_type     = itparam.precond_type;
47      const SHORT stop_type   = itparam.stop_type;
48      const INT   maxit       = itparam.maxit;
49      const REAL  tol         = itparam.tol;
50
51      // Step 1. Get stiffness matrix and right-hand side
52      // Read A and b -- P1 FE discretization for Poisson. The location
53      // of the data files is given in "ini/pcg.dat".
54      dCSRmat A;
55      dvector b, x;
56      char filename1[512], *datafile1;
57      char filename2[512], *datafile2;
58
59      // Read the stiffness matrix from matFE.dat
```

```
60        memcpy(filename1, inparam.workdir, STRLEN);
61        datafile1="csrmat_FE.dat"; strcat(filename1, datafile1);
62
63        // Read the RHS from rhsFE.dat
64        memcpy(filename2, inparam.workdir, STRLEN);
65        datafile2="rhs_FE.dat"; strcat(filename2, datafile2);
66
67        fasp_dcsrvec_read2(filename1, filename2, &A, &b);
68
69        // Step 2. Print problem size and PCG parameters
70        if (print_level>PRINT_NONE) {
71            printf("A: m = %d, n = %d, nnz = %d\n", A.row, A.col, A.nnz);
72            printf("b: n = %d\n", b.row);
73            fasp_param_solver_print(&itparam);
74        }
75
76        // Setp 3. Setup preconditioner
77        // Preconditioner type is determined by pc_type
78        precond *pc = fasp_precond_setup(pc_type, &amgparam, &iluparam, &A);
79
80        // Step 4. Solve the system with PCG as an iterative solver
81        // Set the initial guess to be zero and then solve it using PCG solver
82        // Note that we call PCG interface directly. There is another way which
83        // calls the abstract iterative method interface; see possion-its.c for
84        // more details.
85        fasp_dvec_alloc(A.row, &x);
86        fasp_dvec_set(A.row, &x, 0.0);
87
88        fasp_solver_dcsr_pcg(&A, &b, &x, pc, tol, maxit, stop_type, print_level);
89
90        // Step 5. Clean up memory
91        // First clean up the AMG data if you are using AMG as preconditioner
92        fasp_amg_data_free(((precond_data *)pc->data)->mgl_data, &amgparam);
93
94        // Clean up the pcdata
95        if (pc_type!=PREC_NULL) fasp_mem_free(pc->data);
96        fasp_mem_free(pc);
97
98        // Clean up coefficient matrix, right-hand side, and solution
99        fasp_dcsr_free(&A);
100        fasp_dvec_free(&b);
101        fasp_dvec_free(&x);
102
103        return FASP_SUCCESS;
104 }
105
106 /*--------------------------------*/
107 /*--       End of File         --*/
108 /*--------------------------------*/
```

This example is very similar to the first example, and the details are as follows.

- Line 40–41 sets default parameters. In this example, we need parameters for iterative meth-

ods, AMG preconditioner, and ILU preconditioner.

- Line 77 sets up the desired preconditioner and prepare it for the preconditioned iterative methods.

- Line 87 calls PCG to solve $Ax = b$. One can also call the general iterative method interface as in the previous example.

- Line 90 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_dcsrvec_read2: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec_read2: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969


        Parameters in ITS_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
-----------------------------------------------



Setting up 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 0.0041 seconds.
```

```
Calling CG solver (CSR) ...
_____
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.
```

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-pcg.f90`". Users who would like to call FASP solver from a Fortran based application can see how to do this example.

```fortran
!> \file   poisson-pcg.f90
!>
!> \brief The third test example for FASP: using PCG to solve
!>        the discrete Poisson equation from P1 finite element.
!>        F90 version.
!>
!> \note   Solving the Poisson equation (P1 FEM) with PCG: F90 version
!>
!>------------------------------------------------------------------------------
!>  Copyright (C) 2012--Present by the FASP team. All rights reserved.
!>  Released under the terms of the GNU Lesser General Public License 3.0 or later.
!>------------------------------------------------------------------------------

program test

   implicit none

   double precision, dimension(:), allocatable :: u,b
   double precision, dimension(:), allocatable :: a
   integer,          dimension(:), allocatable :: ia,ja

   integer          :: iufile, n, nnz, i, prt_lvl, maxit
   double precision :: tol

   print*, ""
   write(*,"(A)") "=========================================="
   write(*,"(A)") "||    FASP: PCG example -- F90 version    ||"
   write(*,"(A)") "=========================================="
   print*, ""

   ! Step 0: user defined variables
   prt_lvl = 3
   maxit = 500
   tol = 1.0d-6
   iufile = 1

   ! Step 1: read A and b
```

```fortran
38
39      !===> Read data A from file
40      open(unit=iufile,file='../data/csrmat_FE.dat')
41
42      read(iufile,*) n
43      allocate(ia(1:n+1))
44      read(iufile,*) (ia(i),i=1,n+1)
45
46      nnz=ia(n+1)-ia(1)
47      allocate(ja(1:nnz),a(1:nnz))
48      read(iufile,*) (ja(i),i=1,nnz)
49      read(iufile,*) (a(i),i=1,nnz)
50
51      close(iufile)
52
53      !===> Read data b from file
54      open(unit=iufile,file='../data/rhs_FE.dat')
55
56      read(iufile,*) n
57      allocate(b(1:n))
58      read(iufile,*) (b(i),i=1,n)
59
60      close(iufile)
61
62      !===> Shift the index to start from 0 (for C routines)
63      forall (i=1:n+1) ia(i)=ia(i)-1
64      forall (i=1:nnz) ja(i)=ja(i)-1
65
66      ! Step 2: Solve the system
67
68      !===> Initial guess
69      allocate(u(1:n))
70      u=0.0d0
71      call fasp_fwrapper_dcsr_krylov_amg(n,nnz,ia,ja,a,b,u,tol,maxit,prt_lvl);
72
73      ! Step 3: Clean up memory
74      deallocate(ia,ja,a)
75      deallocate(b,u)
76
77  end program test
78
79  !/*--------------------------------*/
80  !/*--        End of File        --*/
81  !/*--------------------------------*/
```

2.4   Example 4: An GMG solver for the Poisson equation

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 "`tutorial/main/poisson-gmg.c`" and a piece of the source code is listed as follows:

```
1   /*! \file  poisson-gmg.c
2    *
3    *  \brief The fourth test example for FASP: using GMG to solve
4    *         the discrete Poisson equation from five-point finite
5    *         difference stencil. C version.
6    *
7    *  \note  Solving the Poisson equation (FDM) with GMG: C version
8    *
9    *
10   *-----------------------------------------------------------------------------
11   *  Copyright (C) 2013--Present by the FASP team. All rights reserved.
12   *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
13   *-----------------------------------------------------------------------------
14   */
```

```
15
16   #include <time.h>
17   #include <math.h>
18
19   #include "fasp.h"
20   #include "fasp_functs.h"
21
22   const REAL pi = 3.14159265;
23
24   /**
25    * \fn static REAL f2d(INT i, INT j, INT nx, INT ny)
26    *
27    * \brief Setting f in Poisson equation, where
28    *          f = sin(pi x)*sin(pi y)
29    *
30    * \param i      i-th position in x direction
31    * \param j      j-th position in y direction
32    * \param nx     Number of grids in x direction
33    * \param ny     Number of grids in y direction
34    *
35    * \author Ziteng Wang
36    * \date   06/07/2013
37    */
38   static REAL f2d (INT i,
39                    INT j,
40                    INT nx,
41                    INT ny)
42   {
43       return sin(pi *(((REAL) j)/((REAL) ny)))
44              *sin(pi *(((REAL) i)/((REAL) nx)));
45   }
46
47   /**
48    * \fn static REAL L2NormError2d(REAL *u, INT nx, INT ny)
49    *
50    * \brief Computing Discretization Error, where exact solution
51    *          u = sin(pi x)*sin(pi y)/(2*pi*pi)
52    *
53    * \param u      Vector of DOFs
54    * \param nx     Number of grids in x direction
55    * \param ny     Number of grids in y direction
56    *
57    * \author Ziteng Wang
58    * \date   06/07/2013
59    */
60   static REAL L2NormError2d (REAL *u,
61                              INT nx,
62                              INT ny)
63   {
64       const REAL h = 1.0/nx;
65       REAL l2norm  = 0.0, uexact;
66
67       INT i, j;
```

```
68        for ( i = 1; i < ny; i++ ) {
69            for ( j = 1; j < nx; j++ ) {
70                uexact  = sin(pi*i*h)*sin(pi*j*h)/(pi*pi*2.0);
71                l2norm += pow((u[i*(nx+1)+j] - uexact), 2);
72            }
73        }
74
75        return sqrt(l2norm*h*h);
76   }
77
78   /**
79    * \brief An example of GMG method using Full Multigrid cycle
80    *
81    * \author Chensong Zhang
82    * \date    10/12/2015
83    *
84    * \note    Number of grids of nx = ny should be equal to 2^maxlevel.
85    */
86   int main (int argc, const char *argv[])
87   {
88        const REAL rtol   = 1.0e-6;
89        const INT  prtlvl = PRINT_MORE;
90
91        INT        i, j, nx, maxlevel;
92        REAL       *u, *b, h, error0;
93
94        // Step 0. Set number of levels for GMG
95        printf("Enter the desired number of levels:   ");
96        if ( scanf("%d", &maxlevel) > 1 ) {
97            printf("### ERROR: Did not get a valid input !!!\n");
98            return ERROR_INPUT_PAR;
99        }
100
101        // Step 1. Compute right-hand side b and set approximate solution u
102        nx = (int) pow(2.0, maxlevel);
103        h = 1.0/((REAL) nx);
104
105        u = (REAL *)malloc((nx+1)*(nx+1)*sizeof(REAL));
106        fasp_darray_set((nx+1)*(nx+1), u, 0.0);
107
108        b = (REAL *)malloc((nx+1)*(nx+1)*sizeof(REAL));
109        for (i = 0; i <= nx; i++) {
110            for (j = 0; j <= nx; j++) {
111                b[j*(nx+1)+i] = h*h*f2d(i, j, nx, nx);
112            }
113        }
114
115        // Step 2. Solve the Poisson system in 2D with full Multigrid cycle
116        fasp_poisson_fgmg2d(u, b, nx, nx, maxlevel, rtol, prtlvl);
117
118        // Step 3. Compute error in L2 norm
119        error0 = L2NormError2d(u, nx, nx);
120
```

```
121        printf("L2 error ||u-u'|| = %e\n",error0);
122
123        // Step 4. Clean up memory
124        free(u);
125        free(b);
126
127        return FASP_SUCCESS;
128    }
129
130    /*---------------------------------*/
131    /*--          End of File          --*/
132    /*---------------------------------*/
```

2.5   Example 5: Block ILU preconditioner

We now show a simple example for calling iterative solvers in BSR format. The test example is
from a test problem given by the Society of Petroleum Engineers (SPE01 Benchmark) using a fully
implicit black-oil simulator at certain time step. The test matrix is the Jacobian matrix from the
Newton linearization and is stored as a BSR matrix (see §3.2 for details).

The sample code for this solver can be found in "tutorial/main/spe01-its.c" and a piece of
the source code is listed as follows:

```
1    /*! \file  spe01-its.c
2     *
3     *  \brief The fifth test example for FASP: using ITS_BSR to solve
4     *         the Jacobian equation from reservoir simulation benchmark
5     *         problem SPE01.
6     *
7     *  \note  ITS_BSR example for FASP: C version
8     *
9     *  Solving the Society of Petroleum Engineers SPE01 benchmark problem
10    *  with Block ILU preconditioned Krylov methods
11    *
12    *---------------------------------------------------------------------------------
13    *  Copyright (C) 2012--Present by the FASP team. All rights reserved.
14    *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
15    *---------------------------------------------------------------------------------
16    */
17
18   #include "fasp.h"
19   #include "fasp_functs.h"
20
21   /**
22    * \fn int main (int argc, const char * argv[])
23    *
24    * \brief This is the main function for the fourth example.
25    *
26    * \author Feiteng Huang, Chensong Zhang
```

```
27   * \date    05/22/2012
28   *
29   * Modified by Chensong Zhang on 09/22/2012
30   */
31  int main (int argc, const char * argv[])
32  {
33      input_param       inparam;  // parameters from input files
34      ITS_param         itparam;  // parameters for itsolver
35      ILU_param         iluparam; // parameters for ILU
36
37      printf("\n=======================================");
38      printf("\n||   FASP: SPE01 -- ITS BSR version    ||");
39      printf("\n=======================================\n\n");
40
41      // Step 0. Set parameters: We can ini/its_bsr.dat
42      fasp_param_set(argc, argv, &inparam);
43      fasp_param_init(&inparam, &itparam, NULL, &iluparam, NULL);
44
45      // Set local parameters
46      const int print_level = inparam.print_level;
47
48      // Step 1. Get stiffness matrix and right-hand side
49      // Read A and b -- P1 FE discretization for Poisson. The location
50      // of the data files is given in "its.dat".
51      dBSRmat A;
52      dvector b, x;
53      char filename1[512], *datafile1;
54      char filename2[512], *datafile2;
55
56      // Read the stiffness matrix from bsrmat_SPE01.dat
57      memcpy(filename1, inparam.workdir, STRLEN);
58      datafile1="bsrmat_SPE01.dat"; strcat(filename1, datafile1);
59      fasp_dbsr_read(filename1, &A);
60
61      // Read the RHS from rhs_SPE01.dat
62      memcpy(filename2, inparam.workdir, STRLEN);
63      datafile2="rhs_SPE01.dat"; strcat(filename2, datafile2);
64      fasp_dvec_read(filename2, &b);
65
66      // Step 2. Print problem size and ITS_bsr parameters
67      if (print_level>PRINT_NONE) {
68          printf("A: m = %d, n = %d, nnz = %d\n", A.ROW, A.COL, A.NNZ);
69          printf("b: n = %d\n", b.row);
70          fasp_param_solver_print(&itparam);
71          fasp_param_ilu_print(&iluparam);
72      }
73
74      // Step 3. Solve the system with ITS_BSR as an iterative solver
75      // Set the initial guess to be zero and then solve it using standard
76      // iterative methods, without applying any preconditioners
77      fasp_dvec_alloc(b.row, &x);
78      fasp_dvec_set(b.row, &x, 0.0);
79
```

```
80      itparam.itsolver_type = SOLVER_GMRES;
81      fasp_solver_dbsr_krylov_ilu(&A, &b, &x, &itparam, &iluparam);
82
83      // Step 4. Clean up memory
84      fasp_dbsr_free(&A);
85      fasp_dvec_free(&b);
86      fasp_dvec_free(&x);
87
88      return FASP_SUCCESS;
89  }
90
91  /*---------------------------------*/
92  /*--        End of File         --*/
93  /*---------------------------------*/
```

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: SPE01 -- ITS BSR version    ||
====================================


fasp_dbsr_read: reading file ../data/bsrmat_SPE01.dat...
fasp_dvec_read: reading file ../data/rhs_SPE01.dat...
A: m = 302, n = 302, nnz = 1788
b: n = 906


        Parameters in ITS_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
-------------------------------------------------



        Parameters in ILU_param
-------------------------------------------------
ILU print level:              2
ILU type:                     1
ILU level of fill-in:         0
ILU relaxation factor:        0.0000
ILU drop tolerance:           1.00e-03
ILU permutation tolerance:    0.00e+00
-------------------------------------------------


BSR ILU(0) setup costs 0.000185 seconds.

Calling GMRes solver (BSR) ...
-------------------------------------------------
```

```
It Num |    ||r||/||b||    |       ||r||       |  Conv. Factor
_____
      0 |   1.000000e+00   |   8.207069e+03   |      -.---
      1 |   9.999991e-01   |   8.207062e+03   |      1.0000
      2 |   9.991891e-01   |   8.200415e+03   |      0.9992
      3 |   9.984917e-01   |   8.194691e+03   |      0.9993
      4 |   9.581382e-01   |   7.863507e+03   |      0.9596
      5 |   9.387736e-01   |   7.704580e+03   |      0.9798
      6 |   8.996932e-01   |   7.383844e+03   |      0.9584
      7 |   8.970099e-01   |   7.361822e+03   |      0.9970
      8 |   8.570704e-01   |   7.034036e+03   |      0.9555
      9 |   5.309276e-01   |   4.357360e+03   |      0.6195
     10 |   1.462587e-01   |   1.200355e+03   |      0.2755
     11 |   3.520599e-02   |   2.889380e+02   |      0.2407
     12 |   8.488230e-03   |   6.966349e+01   |      0.2411
     13 |   2.019708e-03   |   1.657588e+01   |      0.2379
     14 |   4.524916e-04   |   3.713630e+00   |      0.2240
     15 |   9.670973e-05   |   7.937035e-01   |      0.2137
     16 |   1.970931e-05   |   1.617557e-01   |      0.2038
     17 |   3.905034e-06   |   3.204889e-02   |      0.1981
     18 |   8.553378e-07   |   7.019817e-03   |      0.2190
Number of iterations = 18 with relative residual 8.553446e-07.
Iterative method costs 0.0009 seconds.
ILUk_Krylov method totally costs 0.0011 seconds.
```

2.6   How to change parameters for solvers/preconditioners

In the previous examples, we have seen how to use the default parameters in FASP. In this section
we discuss changing such parameters by reading them from a disk file or from the command line.
An example of parameter initialization file is found in the FASP tutorial directory and is named
"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                 = C       % 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_SWZ_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
60  AMG_smooth_restriction   = ON      % Switch for smoothing restriction or not
```

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 (§3.6).

- Line 56–60 gives the parameters for the setup phase of the aggregation-base AMG methods (§3.6).

You can do a very simple experiment—Simply 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_dcsrvec_read2: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec_read2: 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
_____
```

```
Setting up 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 0.0027 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 0.0060 seconds.
AMG totally costs 0.0090 seconds.
```

You can compare this with the results in §2.1.

Similarly, you can solve the same problem using pairwise unsmoothed aggregation AMG preconditioned conjugate gradient method by calling

```
$ ./poisson-pcg-c.ex -ini ini/amg_ua.dat
```

and it will yield the following result:

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


fasp_dcsrvec_read2: reading file ../data/csrmat_FE.dat...
fasp_dcsrvec_read2: reading file ../data/rhs_FE.dat...
A: m = 3969, n = 3969, nnz = 27281
b: n = 3969


        Parameters in ITS_param
_____

Solver print level:              3
Solver type:                     1
Solver precond type:             2
Solver max num of iter:          1000
Solver tolerance:                1.00e−06
Solver stopping type:            1
_____



Setting up UA AMG ...
_____

   Level    Num of rows    Num of nonzeros    Avg. NNZ / row
_____

     0          3969            27281             6.87
     1          1059             7169             6.77
     2           286             1884             6.59
     3            71              431             6.07
_____

  Grid complexity = 1.357  |  Operator complexity = 1.348
_____

Unsmoothed aggregation setup costs 0.0015 seconds.

Calling CG solver (CSR) ...
_____

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

      0 |   1.000000e+00   |   7.514358e+00   |     —.——
      1 |   5.078363e−01   |   3.816064e+00   |     0.5078
      2 |   8.526434e−02   |   6.407068e−01   |     0.1679
      3 |   3.081067e−02   |   2.315224e−01   |     0.3614
      4 |   7.522033e−03   |   5.652325e−02   |     0.2441
      5 |   1.997295e−03   |   1.500839e−02   |     0.2655
      6 |   5.914181e−04   |   4.444127e−03   |     0.2961
      7 |   1.498444e−04   |   1.125985e−03   |     0.2534
      8 |   4.380269e−05   |   3.291491e−04   |     0.2923
      9 |   1.120054e−05   |   8.416489e−05   |     0.2557
     10 |   2.772669e−06   |   2.083482e−05   |     0.2475
     11 |   8.491093e−07   |   6.380511e−06   |     0.3062
Number of iterations = 11 with relative residual 8.491093e−07.
```

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
```

For example, in order to the change the AMG type to the smoothed aggregation (SA) used by the preconditioner for PCG, you can also use the command line options:

```
./poisson-amg-c.ex -amgtype 2 -amgmaxit 100
```

Here we only changed two parameters from the default setting without changing anything else. So it might not give the same output as in the previous example.

Chapter 3

Data structures and basic usage

In this chapter, we discuss the basic data structures and the important building blocks which are useful 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 data structures most often used for implementing iterative methods are sparse matrices and vectors. In this section, we first discuss the data structures for vectors and matrices in FASP; and then we discuss BLAS operations 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.

```
337       //! off-diagonal entries (dimension is nband * [(ngrid-|offsets|) * nc^2])
338       REAL **offdiag;
339
340  } dSTRmat; /**< Structured matrix of REAL type */
341
342  /**
343   * \struct dvector
344   * \brief  Vector with n entries of REAL type
345   */
346  typedef struct dvector{
347
348      //! number of rows
349      INT row;
```

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.

```
199        //! nonzero entries of A
200        INT *val;
201
202    } iCSRmat; /**< Sparse matrix of INT type in CSR format */
203
204    /**
205     * \struct dCOOmat
206     * \brief  Sparse matrix of REAL type in COO (IJ) format
207     *
208     * Coordinate Format (I,J,A)
209     *
210     * \note The starting index of A is 0.
211     * \note Change I to rowind, J to colind. To avoid with complex.h confliction on I.
212     */
213    typedef struct dCOOmat{
214
215        //! row number of matrix A, m
216        INT row;
217
218        //! column of matrix A, n
219        INT col;
220
221        //! number of nonzero entries
222        INT nnz;
223
224        //! integer array of row indices, the size is nnz
225        INT *rowind;
226
227        //! integer array of column indices, the size is nnz
```

```
228        INT *colind;
```

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:

```
139   * CSR Format (IA,JA,A) in REAL
140   *
141   * \note The starting index of A is 0.
142   */
143   typedef struct dCSRmat{
144
145       //! row number of matrix A, m
146       INT row;
147
```

```
148        //! column of matrix A, n
149        INT col;
150
151        //! number of nonzero entries
152        INT nnz;
153
154        //! integer array of row pointers, the size is m+1
155        INT *IA;
156
157        //! integer array of column indexes, the size is nnz
158        INT *JA;
159
160        //! nonzero entries of A
161        REAL *val;
162
163 #if MULTI_COLOR_ORDER
164        //! color numbers for the adjacency graph of A
165        INT color;
166        //! integer array of row pointers, the size is colors+1
167        INT *IC;
```

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
$$IA = \big\| \, 0 \, \big\| \, 3 \, \big\| \, 7 \, \big\| \, 9 \, \big\| \, 12 \, \big\|$$

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

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

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

NOTE: The indices in JA and entries of val does NOT have to be ordered as seen in this example. Sometimes they are sorted in ascending order in each row. More often, the diagonal entries are stored in the first position in each row and the rest are sorted in ascending order.

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:

$$ IA = \left\| \, 0 \, \right\| \, 2 \, \left\| \, 5 \, \right\| \, 7 \, \left\| \, 10 \, \right\|, $$

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

and

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

NOTE: The CSR format presents challenges to sparse matrix-vector product mainly because of the high cache missing rate due to indirect memory access and irregular access pattern. In order to reduce the cache missing rate, we introduce an improved data format, CSRL.

(iii) CSRL format

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

```
260        //! nonzero entries of A
261        INT *val;
262
263    } iCOOmat; /**< Sparse matrix of INT type in COO format */
264
265    /*!
266     * \struct dCSRLmat
267     * \brief  Sparse matrix of REAL type in CSRL format
268     */
269    typedef struct dCSRLmat{
270
271        //! number of rows
272        INT row;
273
```

```
274        //! number of cols
275        INT col;
276
277        //! number of nonzero entries
278        INT nnz;
279
280        //! number of different values in i-th row, i=0:nrows-1
281        INT dif;
282
283        //! nz_diff[i]: the i-th different value in 'nzrow'
284        INT *nz_diff;
285
286        //! row index of the matrix (length-grouped): rows with same nnz are together
287        INT *index;
288
289        //! j in {start[i],...,start[i+1]-1} means nz_diff[i] nnz in index[j]-row
290        INT *start;
291
292        //! column indices of all the nonzeros
293        INT *ja;
```

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.

```
24  /**
25   * \struct dBSRmat
26   * \brief Block sparse row storage matrix of REAL type
27   *
28   * \note This data structure is adapted from the Intel MKL library. Refer to:
29   * http://software.intel.com/sites/products/documentation/hpc/mkl/lin/index.htm
30   *
31   * \note Some of the following entries are capitalized to stress that they are
32   *       for blocks!
33   */
34  typedef struct dBSRmat {
35
36      //! number of rows of sub-blocks in matrix A, M
37      INT ROW;
38
39      //! number of cols of sub-blocks in matrix A, N
40      INT COL;
41
42      //! number of nonzero sub-blocks in matrix A, NNZ
43      INT NNZ;
44
45      //! dimension of each sub-block
46      INT nb; // NOTE: for the moment, allow nb*nb full block
47
48      //! storage manner for each sub-block
49      INT storage_manner; // 0: row-major order, 1: column-major order
50
51      //! A real array that contains the elements of the non-zero blocks of
52      //! a sparse matrix. The elements are stored block-by-block in row major
53      //! order. A non-zero block is the block that contains at least one non-zero
54      //! element. All elements of non-zero blocks are stored, even if some of
55      //! them is equal to zero. Within each nonzero block elements are stored
56      //! in row-major order and the size is (NNZ*nb*nb).
57      REAL *val;
58
59      //! integer array of row pointers, the size is ROW+1
60      INT *IA;
61
62      //! Element i of the integer array columns is the number of the column in the
63      //! block matrix that contains the i-th non-zero block. The size is NNZ.
64      INT *JA;
65
66  } 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:

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

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

and

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

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) BLC format

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

```
68   /**
69    * \struct dBLCmat
70    * \brief Block REAL CSR matrix format
71    *
72    * \note The starting index of A is 0.
73    */
74   typedef struct dBLCmat {
75
76       //! row number of blocks in A, m
77       INT brow;
78
79       //! column number of blocks A, n
80       INT bcol;
81
82       //! blocks of dCSRmat, point to blocks[brow][bcol]
83       dCSRmat **blocks;
84
85   } dBLCmat; /**< 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/BlaIO.c" and we list the available functions as follows:

```
470
471   FASP_API void topologic_sort_ILU (ILU_data *iludata);
472
473   FASP_API void mulcol_independ_set (AMG_data *mgl,
474                                      INT        gslvl);
475
476
477   /*-------- In file: BlaILUSetupCSR.c --------*/
478
479   FASP_API SHORT fasp_ilu_dcsr_setup (dCSRmat    *A,
480                                       ILU_data   *iludata,
481                                       ILU_param  *iluparam);
482
483
484   /*-------- In file: BlaILUSetupSTR.c --------*/
485
486   FASP_API void fasp_ilu_dstr_setup0 (dSTRmat  *A,
487                                       dSTRmat  *LU);
488
489   FASP_API void fasp_ilu_dstr_setup1 (dSTRmat  *A,
490                                       dSTRmat  *LU);
491
492
493   /*-------- In file: BlaIO.c --------*/
494
495   FASP_API void fasp_dcsrvec_read1(const char* filename, dCSRmat* A, dvector* b);
496
497   FASP_API void fasp_dcsrvec_read2(const char* filemat, const char* filerhs, dCSRmat*↩
          A,
498                                    dvector* b);
499
500   FASP_API void fasp_dcsr_read(const char* filename, dCSRmat* A);
501
502   FASP_API void fasp_dcoo_read(const char* filename, dCSRmat* A);
503
504   FASP_API void fasp_dcoo_read1(const char* filename, dCSRmat* A);
505
506   FASP_API void fasp_dcoovec_bin_read(const char* fni, const char* fnj, const char* ↩
          fna,
507                                       const char* fnb, dCSRmat* A, dvector* b);
508
509   FASP_API void fasp_dcoo_shift_read(const char* filename, dCSRmat* A);
510
511   FASP_API void fasp_dmtx_read(const char* filename, dCSRmat* A);
512
513   FASP_API void fasp_dmtxsym_read(const char* filename, dCSRmat* A);
514
515   FASP_API void fasp_dstr_read(const char* filename, dSTRmat* A);
516
517   FASP_API void fasp_dbsr_read(const char* filename, dBSRmat* A);
518
519   FASP_API void fasp_dvecind_read(const char* filename, dvector* b);
520
```

```c
521  FASP_API void fasp_dvec_read(const char* filename, dvector* b);
522
523  FASP_API void fasp_ivecind_read(const char* filename, ivector* b);
524
525  FASP_API void fasp_ivec_read(const char* filename, ivector* b);
526
527  FASP_API void fasp_dcsrvec_write1(const char* filename, dCSRmat* A, dvector* b);
528
529  FASP_API void fasp_dcsrvec_write2(const char* filemat, const char* filerhs, dCSRmat↩
        * A,
530                                           dvector* b);
531
532  FASP_API void fasp_dcoo_write(const char* filename, dCSRmat* A);
533
534  FASP_API void fasp_dstr_write(const char* filename, dSTRmat* A);
535
536  FASP_API void fasp_dbsr_print(const char* filename, dBSRmat* A);
537
538  FASP_API void fasp_dbsr_write(const char* filename, dBSRmat* A);
539
540  FASP_API void fasp_dvec_write(const char* filename, dvector* vec);
541
542  FASP_API void fasp_dvecind_write(const char* filename, dvector* vec);
543
544  FASP_API void fasp_ivec_write(const char* filename, ivector* vec);
545
546  FASP_API void fasp_dvec_print(const INT n, dvector* u);
547
548  FASP_API void fasp_ivec_print(const INT n, ivector* u);
549
550  FASP_API void fasp_dcsr_print(const dCSRmat* A);
551
552  FASP_API void fasp_dcoo_print(const dCOOmat* A);
553
554  FASP_API void fasp_dbsr_write_coo(const char* filename, const dBSRmat* A);
555
556  FASP_API void fasp_dcsr_write_coo(const char* filename, const dCSRmat* A);
557
558  FASP_API void fasp_dcsr_write_mtx(const char* filename, const dCSRmat* A);
559
560  FASP_API void fasp_dstr_print(const dSTRmat* A);
561
562  FASP_API void fasp_matrix_read(const char* filename, void* A);
563
564  FASP_API void fasp_matrix_read_bin(const char* filename, void* A);
565
566  FASP_API void fasp_matrix_write(const char* filename, void* A, const INT flag);
567
568  FASP_API void fasp_vector_read(const char* filerhs, void* b);
569
570  FASP_API void fasp_vector_write(const char* filerhs, void* b, const INT flag);
571
572  FASP_API void fasp_hb_read(const char* input_file, dCSRmat* A, dvector* b);
```

```
573
574
575  /*-------- In file: BlaOrderingCSR.c --------*/
576
577  FASP_API void fasp_dcsr_CMK_order (const dCSRmat *A,
578                                     INT          *order,
579                                     INT          *oindex);
580
581  FASP_API void fasp_dcsr_RCMK_order (const dCSRmat *A,
```

NOTE: The above function declarations 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 matrix-vector multiplication

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 show the abstract interface for the iterative methods. The following code segment is taken from "base/src/SolCSR.c":

```
37   /**
38    * \fn INT fasp_solver_dcsr_itsolver (dCSRmat *A, dvector *b, dvector *x,
39    *                                    precond *pc, ITS_param *itparam)
40    *
41    * \brief Solve Ax=b by preconditioned Krylov methods for CSR matrices
42    *
43    * \note  This is an abstract interface for iterative methods.
44    *
45    * \param A        Pointer to the coeff matrix in dCSRmat format
46    * \param b        Pointer to the right hand side in dvector format
47    * \param x        Pointer to the approx solution in dvector format
48    * \param pc       Pointer to the preconditioning action
49    * \param itparam  Pointer to parameters for iterative solvers
50    *
51    * \return         Iteration number if converges; ERROR otherwise.
52    *
53    * \author Chensong Zhang
54    * \date   09/25/2009
55    */
56   INT fasp_solver_dcsr_itsolver (dCSRmat    *A,
57                                  dvector    *b,
58                                  dvector    *x,
59                                  precond    *pc,
60                                  ITS_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 Krylov-type iterative methods.

```
607        printf("### DEBUG: matrix size: %d %d %d\n", A->row, A->col, A->nnz);
608        printf("### DEBUG: rhs/sol size: %d %d\n", b->row, x->row);
609   #endif
```

```
610
611        fasp_gettime (&solve_start);
612
613        // ILU setup for whole matrix
614        ILU_data LU;
615        if ( (status = fasp_ilu_dcsr_setup (A, &LU, iluparam)) < 0 ) goto FINISHED;
616
617        // check iludata
618        if ( (status = fasp_mem_iludata_check(&LU)) < 0 ) goto FINISHED;
619
620        // set preconditioner
```

Now we explain this code segment a little bit:

- Line 608–609 performs the setup phase for ILU method. The particular type of ILU method is determined by "`iluparam`"; see §2.6. Line 612 performs a simple memory check for ILU.

- Line 615–617 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 620 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 should now explain the data structure "`itparam`".

```
} ivector; /**< Vector of INT type */

/*---------------------------*/
/*--- Parameter structures --*/
/*---------------------------*/

/**
 * \struct ITS_param
 * \brief  Parameters for iterative solvers
 */
typedef struct {

    SHORT print_level;      /**< print level: 0--10 */
    SHORT itsolver_type;    /**< solver type: see fasp_const.h */
    SHORT decoup_type;      /**< decoupling type */
```

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 safety net */
#define SOLVER_SBiCGstab        12  /**< BiCGstab with safety net */
#define SOLVER_SMinRes          13  /**< MinRes with safety net */
#define SOLVER_SGMRES           14  /**< GMRes with safety net */
#define SOLVER_SVGMRES          15  /**< Variable-restart GMRES with safety net */
#define SOLVER_SVFGMRES         16  /**< Variable-restart FGMRES with safety net */
#define SOLVER_SGCG             17  /**< GCG with safety net */
//----------------------------------------------------------------------------
#define SOLVER_AMG              21  /**< AMG as an iterative solver */
```

3.6   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/SolAMG.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).

```
46   * Modified by Chensong Zhang on 07/26/2014: Add error handling for AMG setup
47   * Modified by Chensong Zhang on 02/01/2021: Add return value
48   */
49  INT fasp_solver_amg(dCSRmat* A, dvector* b, dvector* x, AMG_param* param)
50  {
51      const REAL  tol        = param->tol;
52      const SHORT max_levels = param->max_levels;
53      const SHORT prtlvl     = param->print_level;
54      const SHORT amg_type   = param->AMG_type;
55      const SHORT cycle_type = param->cycle_type;
56      const INT   maxit      = param->maxit;
57      const INT   nnz = A->nnz, m = A->row, n = A->col;
58
59      // local variables
```

```
60        SHORT      status;
61        INT        iter       = 0;
62        AMG_data* mgl         = fasp_amg_data_create(max_levels);
63        REAL       AMG_start = 0, AMG_end;
64
65    #if MULTI_COLOR_ORDER
66        A->color = 0;
67        A->IC    = NULL;
68        A->ICMAP = NULL;
69    #endif
70
71    #if DEBUG_MODE > 0
72        printf("### DEBUG: [-Begin-] %s ...\n", __FUNCTION__);
73    #endif
74
75        if (prtlvl > PRINT_NONE) fasp_gettime(&AMG_start);
76
77        // check matrix data
78        fasp_check_dCSRmat(A);
79
80        // Step 0: initialize mgl[0] with A, b and x
81        mgl[0].A = fasp_dcsr_create(m, n, nnz);
82        fasp_dcsr_cp(A, &mgl[0].A);
83
84        mgl[0].b = fasp_dvec_create(n);
85        fasp_dvec_cp(b, &mgl[0].b);
86
87        mgl[0].x = fasp_dvec_create(n);
88        fasp_dvec_cp(x, &mgl[0].x);
89
90        // Step 1: AMG setup phase
91        switch (amg_type) {
92
93            case SA_AMG: // Smoothed Aggregation AMG setup
94                status = fasp_amg_setup_sa(mgl, param);
95                break;
96
97            case UA_AMG: // Unsmoothed Aggregation AMG setup
98                status = fasp_amg_setup_ua(mgl, param);
99                break;
100
101            default: // Classical AMG setup
102                status = fasp_amg_setup_rs(mgl, param);
103                break;
104        }
105
106        // Step 2: AMG solve phase
107        if (status == FASP_SUCCESS) { // call a multilevel cycle
108
109            switch (cycle_type) {
110
111                case AMLI_CYCLE: // AMLI-cycle
112                    iter = fasp_amg_solve_amli(mgl, param);
```

```
113                    break;
114
115              case NL_AMLI_CYCLE: // Nonlinear AMLI-cycle
116                    iter = fasp_amg_solve_namli(mgl, param);
117                    break;
118
119              default: // V,W-cycles or hybrid cycles (determined by param)
120                    iter = fasp_amg_solve(mgl, param);
121                    break;
122          }
123
124          fasp_dvec_cp(&mgl[0].x, x);
125
126      }
127
128      else { // call a backup solver
129
130          if (prtlvl > PRINT_MIN) {
131              printf("### WARNING: AMG setup failed!\n");
132              printf("### WARNING: Use a backup solver instead!\n");
133          }
134          fasp_solver_dcsr_spgmres(A, b, x, NULL, tol, maxit, 20, 1, prtlvl);
135      }
136
137      // clean-up memory
138      fasp_amg_data_free(mgl, param);
139
140      // print out CPU time if needed
```

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

- Line 51–54 reads some of the parameters from "`AMG_param`", which can be set in an input file; see §2.6.

- Line 72–79 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 82–93 calls three different AMG setup methods, determined by "`amg_type`".

- Line 98–109 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 level solver: 0 iterative |
69                                     % 31 SuperLU | 32 UMFPack | 33 MUMPS | 34 PARDISO
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_SWZ_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                                     % 2 Mofified RS for positive off-diags
94                                     % 3 Compatible Relaxation
95                                     % 4 Aggressive
96  AMG_interpolation_type    = 1      % 1 Direct | 2 Standard | 3 Energy-min
97  AMG_strong_threshold      = 0.3    % Strong threshold
98  AMG_truncation_threshold  = 0.1    % Truncation threshold
99  AMG_max_row_sum           = 0.9    % Max row sum
100
101 %─────────────────────────────────────────────%
102 % parameters for aggregation-type AMG SETUP    %
103 %─────────────────────────────────────────────%
104
105 AMG_aggregation_type      = 2      % 1 Matching | 2 VMB
106 AMG_pair_number           = 2      % Number of pairs in matching
107 AMG_strong_coupled        = 0.08   % Strong coupled threshold
108 AMG_max_aggregation       = 20     % Max size of aggregations
```

```
109   AMG_tentative_smooth     = 0.67   % Smoothing factor for tentative prolongation
110   AMG_smooth_filter        = OFF    % Switch for filtered matrix for smoothing
111   AMG_smooth_restriction   = ON     % Switch for smoothing restriction or not
112   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 features

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 Enabling OpenMP

OpenMP[1] (Open Multiprocessing) is an API that supports multi-platform shared memory multi-processing 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 in FASP, you can simply use the following option during the configuration stage of cmake:

```
$ make config openmp=yes
```

If you use OpenMP very often and do not want type in this extra command-line option, you need to uncomment one line in "FASP.mk". If you do not have "FASP.mk" file, just copy "FASP.mk.example" to "FASP.mk". Then set "openmp" to "yes" on line 44 "FASP.mk":

---

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

59

```
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  #
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 on a Linux or Mac OS X machine, you need to set:

```
$ export OMP_NUM_THREADS=8
```

On a Windows computer, you need to use

```
C:\FASP> set OMP_NUM_THREADS=8
```

in order to use eight threads for the current sessions. In case you need to use eight threads by default, you should set it as a system environment variable.

4.2   Predefined constants

FASP has many predefined constants used in the source files. Using these macros makes the source codes more readable. These constants are defined in "base/include/fasp_const.h" and a printout of this file is below:

```
1  /*! \file  fasp_const.h
2   *
3   *  \brief Definition of FASP constants, including messages, solver types, etc.
4   *
5   *-----------------------------------------------------------------------------
6   *  Copyright (C) 2009--Present by the FASP team. All rights reserved.
7   *  Released under the terms of the GNU Lesser General Public License 3.0 or later.
8   *-----------------------------------------------------------------------------
9   *
10  *  \warning This is for internal use only. Do NOT change!
11  */
12
13  #ifndef __FASP_CONST__            /*-- allow multiple inclusions --*/
14  #define __FASP_CONST__
15
16  /**
17   * \brief Definition of return status and error messages
18   */
```

```
19   #define FASP_SUCCESS             0    /**< return from function successfully */
20   //————————————————————————————————————————————————————————————————————————
21   #define ERROR_READ_FILE         -1    /**< fail to read a file */
22   #define ERROR_OPEN_FILE         -10   /**< fail to open a file */
23   #define ERROR_WRONG_FILE        -11   /**< input contains wrong format */
24   #define ERROR_INPUT_PAR         -13   /**< wrong input argument */
25   #define ERROR_REGRESS           -14   /**< regression test fail */
26   #define ERROR_MAT_SIZE          -15   /**< wrong problem size */
27   #define ERROR_NUM_BLOCKS        -18   /**< wrong number of blocks */
28   #define ERROR_MISC              -19   /**< other error */
29   //————————————————————————————————————————————————————————————————————————
30   #define ERROR_ALLOC_MEM         -20   /**< fail to allocate memory */
31   #define ERROR_DATA_STRUCTURE    -21   /**< problem with data structures */
32   #define ERROR_DATA_ZERODIAG     -22   /**< matrix has zero diagonal entries */
33   #define ERROR_DUMMY_VAR         -23   /**< unexpected input data */
34   //————————————————————————————————————————————————————————————————————————
35   #define ERROR_AMG_INTERP_TYPE   -30   /**< unknown interpolation type */
36   #define ERROR_AMG_SMOOTH_TYPE   -31   /**< unknown smoother type */
37   #define ERROR_AMG_COARSE_TYPE   -32   /**< unknown coarsening type */
38   #define ERROR_AMG_COARSEING     -33   /**< coarsening step failed to complete */
39   #define ERROR_AMG_SETUP         -39   /**< AMG setup failed to complete */
40   //————————————————————————————————————————————————————————————————————————
41   #define ERROR_SOLVER_TYPE       -40   /**< unknown solver type */
42   #define ERROR_SOLVER_PRECTYPE   -41   /**< unknown precond type */
43   #define ERROR_SOLVER_STAG       -42   /**< solver stagnates */
44   #define ERROR_SOLVER_SOLSTAG    -43   /**< solver's solution is too small */
45   #define ERROR_SOLVER_TOLSMALL   -44   /**< solver's tolerance is too small */
46   #define ERROR_SOLVER_ILUSETUP   -45   /**< ILU setup error */
47   #define ERROR_SOLVER_MISC       -46   /**< misc solver error during run time */
48   #define ERROR_SOLVER_MAXIT      -48   /**< maximal iteration number exceeded */
49   #define ERROR_SOLVER_EXIT       -49   /**< solver does not quit successfully */
50   //————————————————————————————————————————————————————————————————————————
51   #define ERROR_QUAD_TYPE         -60   /**< unknown quadrature type */
52   #define ERROR_QUAD_DIM          -61   /**< unsupported quadrature dim */
53   //————————————————————————————————————————————————————————————————————————
54   #define ERROR_LIC_TYPE          -80   /**< wrong license type */
55   //————————————————————————————————————————————————————————————————————————
56   #define ERROR_UNKNOWN           -99   /**< an unknown error type */
57
58   /**
59    * \brief Definition of logic type
60    */
61   #define TRUE                     1    /**< logic TRUE */
62   #define FALSE                    0    /**< logic FALSE */
63
64   /**
65    * \brief Definition of switch
66    */
67   #define ON                       1    /**< turn on certain parameter */
68   #define OFF                      0    /**< turn off certain parameter */
69
70   /**
71    * \brief Print level for all subroutines —— not including DEBUG output
```

```
72   */
73   #define PRINT_NONE              0   /**< silent: no printout at all */
74   #define PRINT_MIN               1   /**< quiet:  print error, important warnings */
75   #define PRINT_SOME              2   /**< some:   print less important warnings */
76   #define PRINT_MORE              4   /**< more:   print some useful debug info */
77   #define PRINT_MOST              8   /**< most:   maximal printouts, no files */
78   #define PRINT_ALL              10   /**< all:    all printouts, including files */
79
80   /**
81    * \brief Definition of matrix format
82    **/
83   #define MAT_FREE                0   /**< matrix-free format: only mxv action */
84   //------------------------------------------------------------------------------
85   #define MAT_CSR                 1   /**< compressed sparse row */
86   #define MAT_BSR                 2   /**< block-wise compressed sparse row */
87   #define MAT_STR                 3   /**< structured sparse matrix */
88   #define MAT_CSRL                6   /**< modified CSR to reduce cache missing */
89   #define MAT_SymCSR              7   /**< symmetric CSR format */
90   #define MAT_BLC                 8   /**< block CSR matrix */
91   //------------------------------------------------------------------------------
92   //     For bordered systems in reservoir simulation
93   //------------------------------------------------------------------------------
94   #define MAT_bCSR               11   /**< block CSR/CSR matrix == 2*2 BLC matrix */
95   #define MAT_bBSR               12   /**< block BSR/CSR matrix */
96   #define MAT_bSTR               13   /**< block STR/CSR matrix */
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 safety net */
113  #define SOLVER_SBiCGstab       12   /**< BiCGstab with safety net */
114  #define SOLVER_SMinRes         13   /**< MinRes with safety net */
115  #define SOLVER_SGMRES          14   /**< GMRes with safety net */
116  #define SOLVER_SVGMRES         15   /**< Variable-restart GMRES with safety net */
117  #define SOLVER_SVFGMRES        16   /**< Variable-restart FGMRES with safety net */
118  #define SOLVER_SGCG            17   /**< GCG with safety 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   /**< Direct Solver: SuperLU */
124  #define SOLVER_UMFPACK         32   /**< Direct Solver: UMFPack */
```

```
125   #define SOLVER_MUMPS            33   /**< Direct Solver: MUMPS */
126   #define SOLVER_PARDISO          34   /**< Direct Solver: PARDISO */
127
128   /**
129    * \brief Definition of iterative solver stopping criteria types
130    */
131   #define STOP_REL_RES             1   /**< relative residual ||r||/||b|| */
132   #define STOP_REL_PRECRES         2   /**< relative B-residual ||r||_B/||b||_B */
133   #define STOP_MOD_REL_RES         3   /**< modified relative residual ||r||/||x|| */
134
135   /**
136    * \brief Definition of preconditioner type for iterative methods
137    */
138   #define PREC_NULL                0   /**< with no precond */
139   #define PREC_DIAG                1   /**< with diagonal precond */
140   #define PREC_AMG                 2   /**< with AMG precond */
141   #define PREC_FMG                 3   /**< with full AMG precond */
142   #define PREC_ILU                 4   /**< with ILU precond */
143   #define PREC_SCHWARZ             5   /**< with Schwarz preconditioner */
144
145   /**
146    * \brief Type of ILU methods
147    */
148   #define ILUk                     1   /**< ILUk */
149   #define ILUt                     2   /**< ILUt */
150   #define ILUtp                    3   /**< ILUtp */
151
152   /**
153    * \brief Type of Schwarz smoother
154    */
155   #define SCHWARZ_FORWARD          1   /**< Forward ordering */
156   #define SCHWARZ_BACKWARD         2   /**< Backward ordering */
157   #define SCHWARZ_SYMMETRIC        3   /**< Symmetric smoother */
158
159   /**
160    * \brief Definition of AMG types
161    */
162   #define CLASSIC_AMG              1   /**< classic AMG */
163   #define SA_AMG                   2   /**< smoothed aggregation AMG */
164   #define UA_AMG                   3   /**< unsmoothed aggregation AMG */
165
166   /**
167    * \brief Definition of aggregation types
168    */
169   #define PAIRWISE                 1   /**< pairwise aggregation, default is SPAIR */
170   #define VMB                      2   /**< VMB aggregation */
171   #define NPAIR                    3   /**< non-symmetric pairwise aggregation */
172   #define SPAIR                    4   /**< symmetric pairwise aggregation */
173
174   /**
175    * \brief Definition of cycle types
176    */
177   #define V_CYCLE                  1   /**< V-cycle */
```

```
178   #define W_CYCLE                 2   /**< W-cycle */
179   #define AMLI_CYCLE              3   /**< AMLI-cycle */
180   #define NL_AMLI_CYCLE           4   /**< Nonlinear AMLI-cycle */
181   #define VW_CYCLE               12   /**< VW-cycle */
182   #define WV_CYCLE               21   /**< WV-cycle */
183
184   /**
185    * \brief Definition of standard smoother types
186    */
187   #define SMOOTHER_JACOBI         1   /**< Jacobi smoother */
188   #define SMOOTHER_GS             2   /**< Gauss-Seidel smoother */
189   #define SMOOTHER_SGS            3   /**< Symmetric Gauss-Seidel smoother */
190   #define SMOOTHER_CG             4   /**< CG as a smoother */
191   #define SMOOTHER_SOR            5   /**< SOR smoother */
192   #define SMOOTHER_SSOR           6   /**< SSOR smoother */
193   #define SMOOTHER_GSOR           7   /**< GS + SOR smoother */
194   #define SMOOTHER_SGSOR          8   /**< SGS + SSOR smoother */
195   #define SMOOTHER_POLY           9   /**< Polynomial smoother */
196   #define SMOOTHER_L1DIAG        10   /**< L1 norm diagonal scaling smoother */
197
198   /**
199    * \brief Definition of specialized smoother types
200    */
201   #define SMOOTHER_BLKOIL        11   /**< Used in monolithic AMG for black-oil */
202   #define SMOOTHER_SPETEN        19   /**< Used in monolithic AMG for black-oil */
203
204   /**
205    * \brief Definition of coarsening types
206    */
207   #define COARSE_RS               1   /**< Classical */
208   #define COARSE_RSP              2   /**< Classical, with positive offdiags */
209   #define COARSE_CR               3   /**< Compatible relaxation */
210   #define COARSE_AC               4   /**< Aggressive coarsening */
211   #define COARSE_MIS              5   /**< Aggressive coarsening based on MIS */
212
213   /**
214    * \brief Definition of interpolation types
215    */
216   #define INTERP_DIR              1   /**< Direct interpolation */
217   #define INTERP_STD              2   /**< Standard interpolation */
218   #define INTERP_ENG              3   /**< Energy minimization interpolation */
219   #define INTERP_EXT              6   /**< Extended interpolation */
220
221   /**
222    * \brief Type of vertices (DOFs) for coarsening
223    */
224   #define GOPT                   -5   /**< Cannot fit in aggregates */
225   #define UNPT                   -1   /**< Undetermined points */
226   #define FGPT                    0   /**< Fine grid points  */
227   #define CGPT                    1   /**< Coarse grid points */
228   #define ISPT                    2   /**< Isolated points */
229
230   /**
```

```
231     * \brief Definition of smoothing order
232     */
233    #define NO_ORDER                   0   /**< Natural order smoothing */
234    #define CF_ORDER                   1   /**< C/F order smoothing */
235
236    /**
237     * \brief Type of ordering for smoothers
238     */
239    #define USERDEFINED                0   /**< User defined order */
240    #define CPFIRST                    1   /**< C-points first order */
241    #define FPFIRST                   -1   /**< F-points first order */
242    #define ASCEND                    12   /**< Ascending order */
243    #define DESCEND                   21   /**< Descending order */
244
245    /**
246     * \brief Some global constants
247     */
248    #define BIGREAL              1e+20   /**< A large real number */
249    #define SMALLREAL            1e-20   /**< A small real number */
250    #define SMALLREAL2           1e-40   /**< An extremely small real number */
251    #define MAX_REFINE_LVL          20   /**< Maximal refinement level */
252    #define MAX_AMG_LVL             20   /**< Maximal AMG coarsening level */
253    #define MIN_CDOF                20   /**< Minimal number of coarsest variables */
254    #define MIN_CRATE              0.9   /**< Minimal coarsening ratio */
255    #define MAX_CRATE             20.0   /**< Maximal coarsening ratio */
256    #define MAX_RESTART             20   /**< Maximal restarting number */
257    #define MAX_STAG                20   /**< Maximal number of stagnation times */
258    #define STAG_RATIO            1e-4   /**< Stagnation tolerance = tol*STAGRATIO */
259    #define FPNA_RATIO            1e-8   /**< Float-point number arithmetic threshold = ↵
           tol*FPNA_RATIO */
260    #define OPENMP_HOLDS          2000   /**< Smallest size for OpenMP version */
261
262    #endif                               /* end if for __FASP_CONST__ */
263
264    /*────────────────────────────────────*/
265    /*──        End of File         ──*/
266    /*────────────────────────────────────*/
```

4.3   Debugging and how to enable it

NOTE: The default FSP build is a RELEASE version (`/O3` or equivalent compiler options are enabled) and such version is compiled with optimization and no warnings are displayed during the build. How to build the FASP library with debugging enabled is described below.

There is a built-in debug feature which is intended to help developers and users to locate malfunctions and bugs in FASP (and hopefully fix them). In order to turn this feature on, you need to add the `debug` 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 run FASP. If you just want to enable the debugging and warnings during the compile stage, you can do so by using

```
$ 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.