

# **SXAMG: a Serial Algebraic Multigrid Solver Library**

VERSION 1.0

Hui Liu

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Overview . . . . .	1
1.2	License . . . . .	1
1.3	Citation . . . . .	1
1.4	Website . . . . .	2
<b>2</b>	<b>Installation</b>	<b>3</b>
2.1	Configuration . . . . .	3
2.2	Options . . . . .	3
2.3	Compilation . . . . .	4
2.4	Installation . . . . .	5
<b>3</b>	<b>Basics</b>	<b>7</b>
3.1	Data Types . . . . .	7
3.2	Matrix . . . . .	7
3.2.1	Matrix Management . . . . .	8
3.3	Vector . . . . .	9
3.3.1	Vector Management . . . . .	9
3.4	BLAS Operations . . . . .	10
3.4.1	Vector . . . . .	10
3.4.2	Matrix-Vector . . . . .	10
3.4.3	Matrix-Matrix . . . . .	10
<b>4</b>	<b>AMG Solver</b>	<b>11</b>
4.1	Data Structures . . . . .	11
4.2	Management . . . . .	13
4.2.1	Initialize Parameters . . . . .	13
4.2.2	Setup . . . . .	13
4.2.3	Solve . . . . .	13
4.2.4	Destroy . . . . .	13
<b>5</b>	<b>Utilities</b>	<b>15</b>
5.1	Print . . . . .	15
5.2	Memory . . . . .	15

5.3	Performance . . . . .	16
<b>6</b>	<b>How to Use</b>	<b>17</b>
6.1	Solver . . . . .	17
6.2	Lower Level Interface . . . . .	18

# Chapter 1

## Introduction

### 1.1 Overview

**SXAMG** is an AMG (Algebraic Multi-Grid) solver library for sparse linear system,  $\mathbf{Ax} = \mathbf{b}$ . The package is designed for Linux, Unix and Mac systems. It is also possible to compile under Windows if `USE_UNIX` is set to 0. The code is written by C, and it is serial.

The initial code is from FASP solver, <http://fasp.sourceforge.net>, which implements a collection of Krylov solvers, AMG solvers and preconditioners.

The purpose of this refactorization is to provide a standalone AMG solver to support other numerical applications and to validate new ideas in the future. The **SXAMG** can serve as a solver and a preconditioning method. **SXAMG** rewrites all the data structures, subroutines, and file structures, and it removes many components of the original implementation and third-party package dependency.

### 1.2 License

The package uses General Public License (GPL) license. If you have any issue, please contact: [hui.sc.liu@gmail.com](mailto:hui.sc.liu@gmail.com)

### 1.3 Citation

The **SXAMG** library can be cited as,

```
@misc{sxamg-library,  
  author="Hui Liu",  
  title="SXAMG: a Serial Algebraic Multigrid Solver Library",
```

```
year="2017",  
note={\url{https://github.com/huijwliu/sxamg/}}  
}
```

## 1.4 Website

The official website for SXAMG is <https://github.com/huijwliu/sxamg/>.

# Chapter 2

## Installation

SXAMG uses `autoconf` and `make` to detect system parameters, to build and to install.

### 2.1 Configuration

The simplest way to configure is to run command:

```
./configure
```

This command will try to find optional packages if applicable and set system parameters.

### 2.2 Options

The script `configure` has many options, if user would like to check, run command:

```
./configure --help
```

Output will be like this,

```
'configure' configures this package to adapt to many kinds of systems.
```

```
Usage: ./configure [OPTION]... [VAR=VALUE]...
```

```
To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE. See below for descriptions of some of the useful variables.
```

```
....
```

```
Optional Features:
```

```
--disable-option-checking ignore unrecognized --enable/--with options
```

```

--disable-FEATURE      do not include FEATURE (same as --enable-FEATURE=no)
--enable-FEATURE[=ARG] include FEATURE [ARG=yes]
--enable-rpath          enable use of rpath (default)
--disable-rpath         disable use of rpath
--with-rpath-flag=FLAG  compiler flag for rpath (e.g., "-Wl,-rpath,")
--disable-assert        turn off assertions
--enable-big-int        use long int for INT
--disable-big-int       use int for INT (default),
--with-int=type         integer type(long|long long)
--enable-long-double    use long double for FLOAT
--disable-long-double   use double for FLOAT (default)

```

Some influential environment variables:

```

CC          C compiler command
CFLAGS      C compiler flags
LDFLAGS     linker flags, e.g. -L<lib dir> if you have libraries in a
            nonstandard directory <lib dir>
LIBS        libraries to pass to the linker, e.g. -l<library>
CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g. -I<include dir> if
            you have headers in a nonstandard directory <include dir>
CXX         C++ compiler command
CXXFLAGS    C++ compiler flags
FC          Fortran compiler command
FCFLAGS     Fortran compiler flags
CPP         C preprocessor
CXXCPP      C++ preprocessor

```

The most important options are,

- `--prefix=PATH` where to install the library, and default directory is `/usr/local/sxamg/`;
- `--enable-rpath` and `--disable-rpath`, use rpath or not, and it is enabled by default;
- `--enable-big-int` and `--disable-big-int`, use big integer or not, and use `int` by default;
- `--with-int=type`, type is `long` or `long long`. This option is checked when big integer is enabled (`--enable-big-int`);
- `--enable-long-double` and `--disable-long-double`, use `long double` or not, and `double` is used by default;

## 2.3 Compilation

After configuration, `Makefile` and related scripts will be set correctly. A simple `make` command can compile the package,

```
make
```



## 2.4 Installation

Run command:

```
make install
```

The package will be installed to a directory. The default is `/usr/local/sxamg/`. A different directory can be set by `--prefix=DIR` when configuring, such as `--prefix=/usr/sxamg/`.



# Chapter 3

## Basics

This chapter introduces basic types, matrix and vector management.

### 3.1 Data Types

SXAMG has two data types, `SX_FLOAT` and `SX_INT`, for floating-point number and integer. They are defined as:

```
#if USE_LONG_DOUBLE
typedef long double      SX_FLOAT;
#else
typedef double          SX_FLOAT;
#endif

#if USE_LONG_LONG
typedef signed long long int  SX_INT;
#elif USE_LONG
typedef signed long int      SX_INT;
#else
typedef signed int          SX_INT;
#endif
```

The macros, `USE_LONG_DOUBLE`, `USE_LONG_LONG` and `USE_LONG`, are set by `configure` to control the real types. User can control through `configure` options.

### 3.2 Matrix

`SX_MAT` defines CSR matrix. The index is from zero. The meanings of the members are clear.

```
typedef struct SX_MAT
```

```
{
    SX_INT num_rows;
    SX_INT num_cols;
    SX_INT num_nnz;

    SX_INT *Ap;
    SX_INT *Aj;
    SX_FLOAT *Ax;
} SX_MAT;
```

## 3.2.1 Matrix Management

### 3.2.1.1 Create

`sx_mat_struct_create` creates the structure of a matrix, and no value is set:

```
SX_MAT sx_mat_struct_create(const SX_INT nrow, const SX_INT ncol, const SX_INT nnz);
```

`sx_mat_create` creates a CSR matrix:

```
SX_MAT sx_mat_create(SX_INT nrow, SX_INT ncol, SX_INT *Ap, SX_INT *Aj, SX_FLOAT *Ax);
```

### 3.2.1.2 Destroy

`sx_mat_destroy` destroys a matrix and frees its memory:

```
void sx_mat_destroy(SX_MAT *A);
```

### 3.2.1.3 Transpose

`sx_mat_trans` gets transpose a matrix:

```
SX_MAT sx_mat_trans(SX_MAT *A);
```

### 3.2.1.4 Sort

`sx_mat_sort` sorts column indices in ascending manner.

```
void sx_mat_sort(SX_MAT *A);
```

## 3.3 Vector

`SX_VEC` defines floating-point vector.  $n$  is the length of the vector, and  $d$  stores the values.

```
typedef struct SX_VEC
{
    SX_INT    n;
    SX_FLOAT *d;
} SX_VEC;
```

### 3.3.1 Vector Management

#### 3.3.1.1 Create

`sx_vec_create` creates a vector of length  $m$ .

```
SX_VEC sx_vec_create(SX_INT m);
```

#### 3.3.1.2 Destroy

`sx_vec_destroy` destroys a vector and frees its memory.

```
void sx_vec_destroy(SX_VEC *u);
```

#### 3.3.1.3 Set Value

`sx_vec_set_value` sets equal value to each component.

```
void sx_vec_set_value(SX_VEC *x, SX_FLOAT val);
```

`sx_vec_set_entry` sets value:  $x[\text{index}] = \text{val}$ ,

```
void sx_vec_set_entry(SX_VEC *x, SX_INT index, SX_FLOAT val);
```

#### 3.3.1.4 Get Value

`sx_vec_get_entry` returns value of  $x[\text{index}]$ ,

```
SX_FLOAT sx_vec_get_entry(SX_VEC *x, SX_INT index);
```

### 3.3.1.5 Copy

`sx_vec_cp` copies source vector to destination vector.

```
void sx_vec_cp(SX_VEC *src, SX_VEC *des);
```

## 3.4 BLAS Operations

### 3.4.1 Vector

`sx_blas_vec_norm2` calculates L2 norm.

```
SX_FLOAT sx_blas_vec_norm2(SX_VEC *x);
```

`sx_blas_vec_dot` calculates dot product.

```
SX_FLOAT sx_blas_vec_dot(SX_VEC *x, SX_VEC *y);
```

`sx_blas_vec_axpyz` computes:  $z = a * x + y$ .

```
void sx_blas_vec_axpyz(const SX_FLOAT a, SX_VEC *x, SX_VEC *y, SX_VEC *z);
```

`sx_blas_vec_axpy` computes:  $y = a * x + y$ .

```
void sx_blas_vec_axpy(const SX_FLOAT a, SX_VEC *x, SX_VEC *y);
```

### 3.4.2 Matrix-Vector

`sx_blas_mat_amxpy` computes:  $y = y + a * A * x$ ,

```
void sx_blas_mat_amxpy(const SX_FLOAT alpha, SX_MAT *A, SX_VEC *x, SX_VEC *y);
```

`sx_blas_mat_mxy` computes:  $y = A * x$ ,

```
void sx_blas_mat_mxy(SX_MAT *A, SX_VEC *x, SX_VEC *y);
```

### 3.4.3 Matrix-Matrix

`sx_blas_mat_rap` returns the matrix-matrix product:  $R * A * P$ ,

```
SX_MAT sx_blas_mat_rap(SX_MAT *R, SX_MAT *A, SX_MAT *P);
```

# Chapter 4

## AMG Solver

### 4.1 Data Structures

[SX\\_SM\\_TYPE](#) defines smoother types. Nine smoothers are implemented.

```
typedef enum
{
    SX_SM_JACOBI      = 1,  /**< Jacobi smoother */
    SX_SM_GS          = 2,  /**< Gauss-Seidel smoother */
    SX_SM_SGS         = 3,  /**< Symmetric Gauss-Seidel smoother */
    SX_SM_SOR         = 4,  /**< SOR smoother */
    SX_SM_SSOR        = 5,  /**< SSOR smoother */
    SX_SM_GSOR        = 6,  /**< GS + SOR smoother */
    SX_SM_SGSOR       = 7,  /**< SGS + SSOR smoother */
    SX_SM_POLY        = 8,  /**< Polynomial smoother */
    SX_SM_L1DIAG      = 9,  /**< L1 norm diagonal scaling smoother */
} SX_SM_TYPE;
```

[SX\\_COARSEN\\_TYPE](#) defines coarsening types, including classical RS coarsening and classical RS coarsening with positive off-diagonals.

```
typedef enum
{
    SX_COARSE_RS      = 1,  /**< Classical */
    SX_COARSE_RSP     = 2,  /**< Classical, with positive offdiags */
} SX_COARSEN_TYPE;
```

[SX\\_INTERP\\_TYPE](#) defines interpolation types, including direct interpolation and standard interpolation.

```
typedef enum
{
    SX_INTERP_DIR     = 1,  /**< Direct interpolation */
}
```

```

    SX_INTERP_STD      = 2,  /**< Standard interpolation */
} SX_INTERP_TYPE;

```

`SX_AMG_PARS` defines AMG parameters. The meaning of each member is explained as comment. For example, `cycle_itr` determines the cycle type: 1 for V-cycle, 2 for W-cycle..

```

typedef struct SX_AMG_PARS
{
    SX_INT verb;

    SX_INT  cycle_itr;      /** type of AMG cycle, 1 is for V, 2 for W */
    SX_FLOAT tol;           /** stopping tolerance for AMG solver */
    SX_FLOAT ctol;         /** stopping tolerance for coarsest solver */
    SX_INT  maxit;          /** maximal number of iterations of AMG */

    SX_COARSEN_TYPE cs_type; /** coarsening type */
    SX_INT max_levels;      /** max number of levels of AMG */
    SX_INT coarse_dof;      /** max number of coarsest level DOF */

    SX_SM_TYPE smoother;   /** smoother type */
    SX_FLOAT relax;         /** relax parseter for SOR smoother */
    SX_INT cf_order;        /** False (0): nature order, True (1): C/F order */
    SX_INT pre_iter;        /** number of presmoothers */
    SX_INT post_iter;       /** number of postsmoothers */
    SX_INT poly_deg;        /** degree of the polynomial smoother */

    SX_INTERP_TYPE interp_type; /** interpolation type */
    SX_FLOAT strong_threshold; /** strong connection threshold for coarsening */
    SX_FLOAT max_row_sum;     /** maximal row sum parseter */
    SX_FLOAT trunc_threshold; /** truncation threshold */
} SX_AMG_PARS;

```

`SX_RTN` is for return values.

```

typedef struct SX_RTN
{
    SX_FLOAT ares;      /* absolute residual */
    SX_FLOAT rres;      /* relative residual */
    SX_INT  nits;       /* number of iterations */
} SX_RTN;

```



## 4.2 Management

### 4.2.1 Initialize Parameters

`sx_amg_pars_init` sets default parameters.

```
void sx_amg_pars_init(SX_AMG_PARS *pars);
```

### 4.2.2 Setup

`sx_amg_setup` setups the hierarchical struture of AMG solver,

```
SX_AMG * sx_amg_setup(SX_MAT *A, SX_AMG_PARS *pars);
```

### 4.2.3 Solve

`sx_solver_amg_solve` solves the linear system using AMG method,

```
SX_RTN sx_solver_amg_solve(SX_AMG *mg, SX_VEC *x, SX_VEC *b);
```

`sx_solver_amg` solves the linear system using AMG method. This function is a high level interface, and user can call it only to solve a linear system, which will setup the AMG solver, solve and destroy the AMG solver.

```
SX_RTN sx_solver_amg(SX_MAT *A, SX_VEC *x, SX_VEC *b, SX_AMG_PARS *pars);
```

### 4.2.4 Destroy

`sx_amg_data_destroy` destroys the AMG object,

```
void sx_amg_data_destroy(SX_AMG **mg);
```



# Chapter 5

## Utilities

### 5.1 Print

`sx_set_log` sets log file. If log file is set, all screen prints will be stored to log file,

```
void sx_set_log(FILE *io);
```

`sx_printf` prints info to screen, if log file is set, it prints to log file and screen,

```
int sx_printf(const char *fmt, ...);
```

### 5.2 Memory

`sx_mem_malloc` allocates memory,

```
void * sx_mem_malloc(size_t size);
```

`sx_mem_calloc` allocates and initializes memory,

```
void * sx_mem_calloc(size_t size, SX_INT type);
```

`sx_mem_realloc` reallocates memory,

```
void * sx_mem_realloc(void *oldmem, size_t tsize);
```

`sx_mem_free` frees memory,

```
void sx_mem_free(void *mem);
```

## 5.3 Performance

`sx_gettime` returns current time stamp, if `t` is not `NULL`, result will be written to `t`,

```
SX_FLOAT sx_gettime(SX_FLOAT *t);
```

# Chapter 6

## How to Use

### 6.1 Solver

This example shows how to use SXAMG as a solver.

```
{
    SX_AMG_PARS pars;
    SX_MAT A;
    SX_VEC b, x;
    int verb = 2;
    SX_RTN rtn;

    /* default pars */
    sx_amg_pars_init(&pars);

    /* redefine parameters */
    pars.maxit = 1000;
    pars.verb = 2;

    /* set A, b, initial x */
    ....

    /* solve the system */
    rtn = sx_solver_amg(&A, &x, &b, &pars);

    /* free memory */
    sx_mat_destroy(&A);
    sx_vec_destroy(&b);
    sx_vec_destroy(&x);
}
```

We can see that only two functions are called: 1) `sx_amg_pars_init` initializes default AMG parameters; 2) `sx_solver_amg` setups, solves and destroys the AMG system, and the solution is stored by `x`.

The user needs to provide, matrix, right-hand side and initial guess.

## 6.2 Lower Level Interface

This example exposes more details of the AMG solver: 1) setup; 2) solve; 3) destroy.

```
{
    SX_AMG_PARS pars;
    SX_MAT A;
    SX_VEC b, x;
    int verb = 2;
    SX_AMG *mg;
    SX_RTN rtn;

    /* pars */
    sx_amg_pars_init(&pars);
    pars.maxit = 1000;
    pars.verb = 2;

    /* input A */
    ....

    // Step 1: AMG setup phase
    mg = sx_amg_setup(&A, &pars);

    // Step 2: AMG solve phase
    /* rhs and initial guess */
    x = sx_vec_create(A.num_rows);
    sx_vec_set_value(&x, 1.0);

    b = sx_vec_create(A.num_rows);
    sx_vec_set_value(&b, 1.0);

    /* solve */
    rtn = sx_solver_amg_solve(mg, &x, &b);

    sx_mat_destroy(&A);
    sx_vec_destroy(&b);
    sx_vec_destroy(&x);
    sx_amg_data_destroy(&mg);
}
```

The user can modify this example to use SXAMG as a preconditioner, where the AMG system is assembled once and other solvers can call solving phase in each iteration.