LSSP: a Linear Solver Package for Sparse Matrix

VERSION 1.0

Hui Liu

Contents

Chapter 1

Introduction

1.1 Overview

LSSP is a **L**inear **S**olver library for **SP**arse linear system, `Ax = b`. The package is designed for Linux, Unix and Mac systems. It is also possible to compile under Windows. The code is written by C++ and C, and it is serial.

LSSP has implemented many solvers and preconditioners, and it also has interfaces to external packages, such as PETSc, MUMPS, FASP, UMFPACK (SUITESPARSE), KLU (SUITESPARSE), LASPACK, ITSOL, LIS, QR_MUMPS, PARDISO, SUPERLU, and HSL MI20.

1.2 Cite

If you like our LSSP library, you may cite it like this,

```
@misc{lssp-library,
    author="Hui Liu",
    title="LSSP: a Linear Solver Package for Sparse Matrix",
    year="2019",
    note={\url{https://github.com/huiscliu/lssp/}}
}
```

1.3 Website

The official website for LSSP is https://github.com/huiscliu/lssp/.

Chapter 2

Installation

LSSP has interfaces to some external packages, such as PETSc, MUMPS, FASP, UMFPACK (SuiteSparse), KLU (SuiteSparse), LASPACK, ITSOL, LIS, QR_MUMPS, PARDISO, SUPERLU, and HSL MI20. All these packages are optional. Most packages are enabled by default. However, they will disabled if not found by configuration script.

LSSP uses `autoconf` and `make` to detect these packages and 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 from certain directories. Searching details can be read from configure.in and some are explained below. It also sets 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.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print `checking ...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for `--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or `..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local/lssp]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, `make install' will install all the files in
`/usr/local/lssp/bin', `/usr/local/lssp/lib' etc.  You can specify
an installation prefix other than `/usr/local/lssp' using `--prefix',
for instance `--prefix=HOME'.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR            user executables [EPREFIX/bin]
  --sbindir=DIR           system admin executables [EPREFIX/sbin]
  --libexecdir=DIR        program executables [EPREFIX/libexec]
  --sysconfdir=DIR        read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR     modifiable single-machine data [PREFIX/var]
  --libdir=DIR            object code libraries [EPREFIX/lib]
  --includedir=DIR        C header files [PREFIX/include]
  --oldincludedir=DIR     C header files for non-gcc [/usr/include]
  --datarootdir=DIR       read-only arch.-independent data root [PREFIX/share]
  --datadir=DIR           read-only architecture-independent data [DATAROOTDIR]
  --infodir=DIR           info documentation [DATAROOTDIR/info]
  --localedir=DIR         locale-dependent data [DATAROOTDIR/locale]
  --mandir=DIR            man documentation [DATAROOTDIR/man]
  --docdir=DIR            documentation root [DATAROOTDIR/doc/PACKAGE]
  --htmldir=DIR           html documentation [DOCDIR]
```

```
  --dvidir=DIR            dvi documentation [DOCDIR]
  --pdfdir=DIR            pdf documentation [DOCDIR]
  --psdir=DIR             ps documentation [DOCDIR]

System types:
  --build=BUILD     configure for building on BUILD [guessed]
  --host=HOST       cross-compile to build programs to run on HOST [BUILD]

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-omp           enable OpenMP
  --disable-omp          disable OpenMP (default)
  --with-omp-flags=FLAGS compiler flags for OpenMP
  --disable-openmp        do not use OpenMP
  --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-opt         enable OPTION support (default)
  --disable-opt        disable OPTION support
  --enable-blas         enable BLAS support (default)
  --disable-blas        disable BLAS support
  --with-blas=blas BLAS lib
  --enable-lapack         enable LAPACK support (default)
  --disable-lapack        disable LAPACK support
  --with-lapack=lapack LAPACK lib
  --enable-laspack      enable LASPACK support (default)
  --disable-laspack     disable LASPACK support
  --with-laspack-libdir=DIR path for LASPACK library
  --with-laspack-incdir=DIR path for LASPACK header file
  --enable-ssparse      enable SSPARSE support (default)
  --disable-ssparse     disable SSPARSE support
  --with-ssparse-libdir=DIR path for SSPARSE library
  --with-ssparse-incdir=DIR path for SSPARSE header file
  --enable-mumps        enable MUMPS solver (default)
  --disable-mumps       disable MUMPS solver
  --with-mumps-incdir=DIR MUMPS header files directory
  --with-mumps-libdir=DIR MUMPS libraries directory
  --enable-petsc        enable PETSC solver (default)
  --disable-petsc       disable PETSC solver
  --with-petsc-incdir=DIR PETSC header files directory
  --with-petsc-libdir=DIR PETSC libraries directory
  --enable-itsol      enable ITSOL support (default)
  --disable-itsol     disable ITSOL support
  --with-itsol-libdir=DIR path for ITSOL library
```

```
  --with-itsol-incdir=DIR path for ITSOL header file
  --enable-lis      enable LIS support (default)
  --disable-lis     disable LIS support
  --with-lis-libdir=DIR path for LIS library
  --with-lis-incdir=DIR path for LIS header file
  --enable-qrmumps      enable QR_MUMPS support (default)
  --disable-qrmumps     disable QR_MUMPS support
  --with-qrmumps-libdir=DIR path for QR_MUMPS library
  --with-qrmumps-incdir=DIR path for QR_MUMPS header file
  --enable-fasp      enable FASP support (default)
  --disable-fasp     disable FASP support
  --with-fasp-libdir=DIR path for FASP library
  --with-fasp-incdir=DIR path for FASP header file
  --enable-superlu      enable SUPERLU support (default)
  --disable-superlu     disable SUPERLU support
  --with-superlu-libdir=DIR path for SUPERLU library
  --with-superlu-incdir=DIR path for SUPERLU header file
  --enable-pardiso      enable PARDISO support (default)
  --disable-pardiso     disable PARDISO support
  --with-pardiso-libdir=DIR path for PARDISO library
  --with-pardiso-incdir=DIR path for PARDISO header file
  --enable-hslmi20      enable HSL_MI20 support (default)
  --disable-hslmi20     disable HSL_MI20 support
  --with-hslmi20-libdir=DIR path for HSL_MI20 library
  --with-hslmi20-incdir=DIR path for HSL_MI20 header file

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

Use these variables to override the choices made by ‘configure’ or to help
it to find libraries and programs with nonstandard names/locations.
```

The options follow the same convention,

- `--enable-pack`, to enable package `pack`, such as `--enable-itsol`;

- `--disable-pack`, to disable package `pack`, such as `--disable-itsol`;
- `--with-pack-libdir=DIR`, to set `DIR` as the library path of package `pack`, such as `--with-itsol-libdir=/usr/local/itsol/lib/`;
- `--with-pack-incdir=DIR`, to set `DIR` as the include path of package `pack`, such as `--with-itsol-incdir=/usr/local/itsol/include/`;

The configuration script tries to find package from `/usr/local/`, and `/opt/`, such as `/usr/local/itsol/`, and it tries to set correct include path, library path, and specific libraries. However, if configure cannot find correct information, users can help configure by using options.

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/lssp/. A different directory can be set by `--prefix=DIR`.

2.5 Optional Packages

2.5.1 BLAS

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. The Level 1 BLAS perform scalar, vector and vector-vector operations, the Level 2 BLAS perform matrix-vector operations, and the Level 3 BLAS perform matrix-matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example.

Official website: http://www.netlib.org/blas/

BLAS search directories:

```
/usr/local/lib
/usr/local/lib64
/usr/local/blas/lib
/usr/local/blas*/lib
/usr/local/blas/
/usr/local/blas*/
/usr/lib
/usr/lib64
/opt/blas/lib
/opt/blas*/lib
```

2.5.2 LAPACK

LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

Official website: http://www.netlib.org/lapack/

LAPACK search directories:

```
/usr/local/lib
/usr/local/lib64
/usr/local/lapack/lib
/usr/local/lapack*/lib
/usr/local/lapack/
/usr/local/lapack*/
/usr/lib
/usr/lib64
/opt/lapack/lib
/opt/lapack*/lib
```

2.5.3 LASPack

LASPack is a package for solving large sparse systems of linear equations like those which arise from discretization of partial differential equations. It contains classical as well as selected state-of-the-art algorithms which are commonly used for large sparse systems such as CG-like methods for non-symmetric systems (CGN, GMRES, BiCG, QMR, CGS, and BiCGStab) and multilevel methods such as multigrid and conjugate gradient method preconditioned by multigrid and BPX preconditioners. LASPack is written in ANSI C and is thus largely portable.

Official website: http://www.netlib.org/utk/misc/sw_survey/urc/html/LASPack.1.html

LASPack search directories (include and lib):

```
/usr/local/laspack/
/usr/local/laspack*/
/usr/local/
/usr/
/opt/laspack/
/opt/laspack*
```

The include and lib directories are sub-directories of above directories, such as /usr/local/laspack/include and /usr/local/laspack/lib. Users can also set customized directories with options: –with-laspack-libdir=DIR and –with-laspack-incdir=DIR.

2.5.4 SuiteSparse

A Suite of Sparse matrix software, including UMFPACK, CHOLMOD, SPQR, KLU, BTF, and ordering methods (AMD, CAMD, COLAMD, and CCOLAMD).

Official website: https://github.com/jluttine/suitesparse

Official website: http://www.suitesparse.com

SuiteSparse search directories:

```
/usr/local/SuiteSparse
/usr/local/SuiteSparse*
/usr/local
/usr
/opt/SuiteSparse
/opt/SuiteSparse*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.5 MUMPS

MUMPS (MUltifrontal Massively Parallel sparse direct Solver) is a software application for the solution of large sparse systems of linear algebraic equations on distributed memory parallel computers. It was developed in European project PARASOL (1996–1999) by CERFACS, IRIT-ENSEEIHT and RAL. The software implements the multifrontal method, which is a version of Gaussian elimination for large sparse systems of equations, especially those arising from the finite element method. It is written in Fortran 90 with parallelism by MPI and it uses BLAS and ScaLAPACK kernels for dense matrix computations. Since 1999, MUMPS has been supported by CERFACS, IRIT-ENSEEIHT, and INRIA.

Official website: http://mumps.enseeiht.fr/

MUMPS search directories:

```
/usr/local/mumps-seq/
/usr/local/mumps*-seq/
/usr/local/
/usr/
/opt/mumps-seq/
/opt/mumps*-seq/
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.6 PETSC

PETSc, pronounced PET-see (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc is intended for use in large-scale application projects, many ongoing computational science projects are built around the PETSc libraries. PETSc is easy to use for beginners. Moreover, its careful design allows advanced users to have detailed control over the solution process. PETSc includes a large suite of parallel linear, nonlinear equation solvers and ODE integrators that are easily used in application codes written in C, C++, Fortran and now Python. PETSc provides many of the mechanisms needed within parallel application codes, such as simple parallel matrix and vector assembly routines that allow the overlap of communication and computation. In addition, PETSc includes support for parallel distributed arrays useful for finite difference methods.

Official website: https://www.mcs.anl.gov/petsc/

PETSC search directories:

```
/usr/local/petsc-seq
/usr/local/petsc*-seq/
/opt/petsc-seq/
/opt/petsc*-seq/
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.7 ITSOL

TSOL is a library of iterative solvers for general sparse linear systems of equations. ITSOL can be viewed as an extension of the itsol module in SPARSKIT. It is written in C and offers a selection of recently developed preconditioners. The preconditioner suite includes: ILUK, ILUT, ILUC, VBILUK, VBILUT, and ARMS.

Official website: http://www-users.cs.umn.edu/~saad/software/ITSOL/index.html

ITSOL search directories:

```
/usr/local/itsol
/usr/local/itsol*
/usr/local
/usr
/opt/itsol
/opt/itsol*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.8 LIS

Lis (Library of Iterative Solvers for linear systems, pronounced [lis]) is a parallel software library for solving linear equations and eigenvalue problems that arise in the numerical solution of partial differential equations using iterative methods.

Official website: http://www.ssisc.org/lis/

LIS search directories:

```
/usr/local/lis
/usr/local/lis*
/usr/local
/usr
/opt/lis
/opt/lis*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.9 QR_MUMPS

QR_MUMPS is a software package for the solution of sparse, linear systems on multicore computers. It implements a direct solution method based on the QR factorization of the input matrix. Therefore, it is suited to solving sparse least-squares problems and to computing the minimum-norm solution of sparse, underdetermined problems. It can obviously be used for solving square problems in which case the stability provided by the use of orthogonal transformations comes at the cost of a higher operation count with respect to solvers based on, e.g., the LU factorization. It supports real and complex, single or double precision arithmetic.

Official website: http://buttari.perso.enseeiht.fr/qr_mumps/

QR_MUMPS search directories:

```
/usr/local/qr_mumps
/usr/local/qr_mumps*
/usr/local
/usr
/opt/qr_mumps
/opt/qr_mumps*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.10 FASP

FASP team plans to construct a pool of discrete problems arising from partial differential equations (PDEs) or PDE systems and efficient linear solvers for these problems. They mainly utilize the methodology of Auxiliary Space Preconditioning (ASP) to construct efficient linear solvers. A set of Krylov solvers and AMG solvers have been implemented.

Official website: http://fasp.sourceforge.net/

FASP search directories:

```
/usr/local/fasp
/usr/local/fasp*
/usr/local
/usr
/opt/fasp
/opt/fasp*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.11 SUPERLU

SuperLU is a general purpose library for the direct solution of large, sparse, nonsymmetric systems of linear equations on high performance machines. The library is written in C and is callable from either C or Fortran. The library routines will perform an LU decomposition with partial pivoting and triangular system solves through forward and back substitution. The LU factorization routines can handle non-square matrices but the triangular solves are performed only for square matrices.

Official website: http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

SUPERLU search directories:

```
/usr/local/superlu
/usr/local/superlu*
/usr/local
/usr
/opt/superlu
/opt/superlu*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories with options.

2.5.12 PARDISO

The package PARDISO is a thread-safe, high-performance, robust, memory efficient and easy to use software for solving large sparse symmetric and unsymmetric linear systems of equations on shared-memory and distributed-memory multiprocessors.

Official website: http://www.pardiso-project.org/

PARDISO search directories:

```
/usr/local/intel
/opt/intel

/usr/local/pardiso
/usr/local/pardiso*
/usr/local/lib
/usr/local/lib64
/usr/lib
/usr/lib64
/opt/pardiso
/opt/pardiso*
```

2.5.13 HSL MI20 (AMG)

An AMG package using classical method.

Official website: http://www.hsl.rl.ac.uk/catalogue/hsl_mi20.html

HSL MI20 search directories:

```
/usr/local/hsl_mi20
/usr/local/hsl_mi20*
/opt/hsl_mi20
/opt/hsl_mi20*
```

2.5.14 SXAMG

An AMG package using classical method.

Official website: https://github.com/huiscliu/sxamg/

SXAMG search directories:

```
/usr/local/sxamg
/usr/local/sxamg*
/opt/sxamg
/opt/sxamg*
```

The include and lib directories are sub-directories of above directories. Users can also set customized directories using options.

Chapter 3

Matrix and Vector

3.1 Matrix Definition

LSSP uses `int` for integer and `double` for floating-point number. In this library, matrix indices and array indices follow C style, which start from 0. Three matrix types are defined, include `lssp_mat_csr`, `lssp_mat_coo`, and `lssp_mat_bcsr`, which are CSR matrix, COO matrix and block-wise CSR matrix. They are defined as,

```
typedef struct lssp_mat_csr_
{
    int num_rows;
    int num_cols;
    int num_nnzs;
    int *Ap;
    int *Aj;
    double *Ax;

} lssp_mat_csr;
```

```
typedef struct lssp_mat_coo_
{
    int num_rows;
    int num_cols;
    int num_nnzs;
    int *Ai;
    int *Aj;
    double *Ax;

} lssp_mat_coo;
```

```
typedef struct lssp_mat_bcsr_
```

```
{
    int num_rows;
    int num_cols;
    int num_nnzs;
    int blk_size;
    int *Ap;
    int *Aj;
    double *Ax;

} lssp_mat_bcsr;
```

The definitions of `lssp_mat_csr` and `lssp_mat_coo` are the same as standard definitions. For `lssp_mat_bcsr`, each block has the same size, which is defined by `blk_size`. It uses column-major style as `Fortran`.

3.2 Matrix Management

3.2.1 Initialize

`lssp_mat_init` initializes a matrix, which set row, column and non-zero to zero and set arrays to `NULL`.

```
void lssp_mat_init(lssp_mat_csr &A);
```

```
void lssp_mat_init(lssp_mat_coo &A);
```

```
void lssp_mat_init(lssp_mat_bcsr &A);
```

3.2.2 Create

`lssp_mat_create` creates a CSR matrix.

```
lssp_mat_csr lssp_mat_create(int nrows, int ncols, int *Ap, int *Aj, double *Ax);
```

3.2.3 Destroy

`lssp_mat_destroy` destroys a matrix object and releases memory.

```
void lssp_mat_destroy(lssp_mat_csr &csr);
```

```
void lssp_mat_destroy(lssp_mat_coo &coo);
```

```
void lssp_mat_destroy(lssp_mat_bcsr &csr);
```

3.2.4 Conversion

`lssp_mat_csr_to_bcsr` converts a CSR matrix to block CSR matrix.

```
lssp_mat_bcsr lssp_mat_csr_to_bcsr(const lssp_mat_csr A, int bs);
```

`lssp_mat_bcsr_to_csr` converts a block CSR matrix to a CSR matrix.

```
lssp_mat_csr lssp_mat_bcsr_to_csr(const lssp_mat_bcsr A);
```

`lssp_mat_csr_to_coo` converts a CSR matrix to a COO matrix.

```
lssp_mat_coo lssp_mat_csr_to_coo(const lssp_mat_csr A);
```

`lssp_mat_coo_to_csr` converts a COO matrix to CSR matrix.

```
lssp_mat_csr lssp_mat_coo_to_csr(const lssp_mat_coo coo);
```

`lssp_mat_transpose`: CSR matrix transpose.

```
lssp_mat_csr lssp_mat_transpose(const lssp_mat_csr A);
```

3.2.5 Sort

`lssp_mat_sort_column` sorts a CSR matrix in ascending order.

```
void lssp_mat_sort_column(lssp_mat_csr &A);
```

`lssp_mat_csr_is_sorted` checks if a CSR matrix is sorted.

```
bool lssp_mat_csr_is_sorted(const lssp_mat_csr A);
```

`lssp_mat_bcsr_is_sorted` checks if a block CSR matrix is sorted.

```
bool lssp_mat_bcsr_is_sorted(const lssp_mat_bcsr A);
```

`lssp_mat_adjust_zero_diag` changes zero diagonal element to some small value.

```
lssp_mat_csr lssp_mat_adjust_zero_diag(const lssp_mat_csr A, double tol);
```

3.3 Vector Definition

```
typedef struct lssp_vec_
{
    int n;
    double *d;

} lssp_vec;
```

`lssp_vec` has two members, which are vector length (`n`) and data (memory, `d`).

3.4 Vector Management

3.4.1 Create

`lssp_vec_create` creates a length `n` floating-point vector.

```
lssp_vec lssp_vec_create(int n);
```

3.4.2 Destroy

`lssp_vec_destroy` destroys a vector.

```
void lssp_vec_destroy(lssp_vec &v);
```

3.4.3 Set Value

`lssp_vec_set_value`, `lssp_vec_set_value_by_array` and `lssp_vec_set_value_by_index` set vector values.

`lssp_vec_set_value` sets the vector to the same value.

```
void lssp_vec_set_value(lssp_vec x, double val);
```

`lssp_vec_set_value_by_array` sets vector value by a buffer, which has the same length as vector.

```
void lssp_vec_set_value_by_array(lssp_vec x, double *val);
```

`lssp_vec_set_value_by_index` sets value to the i-th component.

```
void lssp_vec_set_value_by_index(lssp_vec x, int i, double val);
```

3.4.4 Get Value

lssp_vec_get_value copies vector's values to a buffer, which should have the same length as the vector.

```
void lssp_vec_get_value(double *val, lssp_vec x);
```

lssp_vec_get_value_by_index gets the value of the i-th component.

```
double lssp_vec_get_value_by_index(lssp_vec x, int i);
```

3.4.5 Copy

lssp_vec_copy copies data from source to destination.

```
void lssp_vec_copy(lssp_vec des, const lssp_vec src);
```

3.5 BLAS 1

lssp_vec_axpby computes: $y = \beta * y + \alpha * x$.

```
void lssp_vec_axpby(double alpha, const lssp_vec x, double beta, lssp_vec y);
```

lssp_vec_axpbyz computes: $z = \beta * y + \alpha * x$.

```
void lssp_vec_axpbyz(double alpha, const lssp_vec x, double beta,
    lssp_vec y, lssp_vec z);
```

lssp_vec_dot computes dot product.

```
double lssp_vec_dot(const lssp_vec x, const lssp_vec y);
```

lssp_vec_norm computes L2 norm.

```
double lssp_vec_norm(const lssp_vec x);
```

lssp_vec_scale scales a vector.

```
void lssp_vec_scale(lssp_vec x, double a);
```

3.6 BLAS 2

lssp_mv_amxpby computes: $y = \beta * y + \alpha * A * x$.

```
void lssp_mv_amxpby(double alpha, const lssp_mat_csr A, const lssp_vec x, \
    double beta, lssp_vec y);
```

lssp_mv_amxpbyz computes: $z = \beta * y + \alpha * A * x$.

```
void lssp_mv_amxpbyz(double alpha, const lssp_mat_csr A, const lssp_vec x, \
    double beta, const lssp_vec y, lssp_vec z);
```

lssp_mv_amxy computes: $y = a * A * x$.

```
void lssp_mv_amxy(double a, const lssp_mat_csr A, const lssp_vec x, lssp_vec y);
```

lssp_mv_mxy computes: $y = A * x$.

```
void lssp_mv_mxy(const lssp_mat_csr A, const lssp_vec x,  lssp_vec y);
```

Chapter 4

Linear Solvers and Preconditioners

4.1 Solver Types

The solver type is defined as LSSP_SOLVER_TYPE,

```
/* solver type */
typedef enum LSSP_SOLVER_TYPE_
{
    LSSP_SOLVER_GMRES,     /* gmres(m) */
    LSSP_SOLVER_LGMRES,    /* lgmres(m, k) */
    LSSP_SOLVER_BICGSTAB,  /* bicgstab */
    LSSP_SOLVER_BICGSTABL, /* bicgstab(l) */
    LSSP_SOLVER_BICGSAFE,  /* bicgsafe */
    LSSP_SOLVER_CG,        /* cg */
    LSSP_SOLVER_CGS,       /* cgs */
    LSSP_SOLVER_GPBICG,    /* gpbicg */
    LSSP_SOLVER_CR,        /* cr */
    LSSP_SOLVER_CRS,       /* crs */
    LSSP_SOLVER_BICRSTAB,  /* bicrstab */
    LSSP_SOLVER_BICRSAFE,  /* bicrsafe */
    LSSP_SOLVER_GPBICR,    /* gpbicr */
    LSSP_SOLVER_QMRCGSTAB, /* qmrcgstab */
    LSSP_SOLVER_TFQMR,     /* tfqmr */
    LSSP_SOLVER_ORTHOMIN,  /* orthomin */
    LSSP_SOLVER_IDRS,      /* idrs */

#if USE_LASPACK
    LSSP_SOLVER_LASPACK,   /* laspack */
#endif

#if USE_SSPARSE
    LSSP_SOLVER_UMFPACK,   /* ssparse, umf */
```

```
    LSSP_SOLVER_KLU,        /* ssparse, klu */
#endif

#if USE_MUMPS
    LSSP_SOLVER_MUMPS,      /* mumps */
#endif

#if USE_PETSC
    LSSP_SOLVER_PETSC,      /* petsc */
#endif

#if USE_LIS
    LSSP_SOLVER_LIS,        /* lis */
#endif

#if USE_QR_MUMPS
    LSSP_SOLVER_QR_MUMPS,   /* qr-mumps */
#endif

#if USE_FASP
    LSSP_SOLVER_FASP,       /* FASP */
    LSSP_SOLVER_AMG,        /* amg from FASP */
    LSSP_SOLVER_FMG,        /* fmg from FASP */
#endif

#if USE_SUPERLU
    LSSP_SOLVER_SUPERLU,    /* superlu */
#endif

#if USE_PARDISO
    LSSP_SOLVER_PARDISO,    /* pardiso */
#endif

#if USE_HSL_MI20
    LSSP_SOLVER_MI20AMG,    /* MI20 AMG */
#endif

#if USE_SXAMG
    LSSP_SOLVER_SXAMG,      /* SXAMG */
#endif

} LSSP_SOLVER_TYPE;
```

LSSP implements many solvers, including `GMRES`, `LGMRES`, `BICGSTAB`, `BICGSTABL`, `BICGSAFE`, `CG`, `CGS`, `GPBICG`, `CR`, `CRS`, `BICRSTAB`, `BICRSAFE`, `GPBICR`, `QMRCGSTAB`, `TFQMR`, `ORTHOMIN`, and `IDRS`.

LSSP also implements interfaces to other famous linear solver packages, such as `LASPACK`, `SSPARSE`, `MUMPS`, `PETSC`, `LIS`, `QR_MUMPS`, `FASP`, `SUPERLU`, `PARDISO`, `HSL_MI20` and `SXAMG`.

4.2 Preconditioner Types

```
typedef enum LSSP_PC_TYPE_
{
    LSSP_PC_NON,
    LSSP_PC_ILUK,               /* ILUK */
    LSSP_PC_ILUT,               /* ILUT */

#if USE_BLAS
#if USE_LAPACK
    LSSP_PC_BILUK,              /* block-wise ILUK */
#endif
#endif

#if USE_ITSOL
    LSSP_PC_BILUT,              /* block-wise ILUT */
    LSSP_PC_VBILUT,             /* var block-wise ILUT */
    LSSP_PC_VBILUK,             /* var block-wise ILUK */
    LSSP_PC_ARMS,               /* multi-level */
#endif

#if USE_FASP
    LSSP_PC_AMG,                /* AMG from FASP */
    LSSP_PC_FMG,                /* FMG from FASP */
#endif

#if USE_HSL_MI20
    LSSP_PC_MI20AMG,            /* AMG from HSL MI20 */
#endif

#if USE_SXAMG
    LSSP_PC_SXAMG,              /* AMG from SXAMG */
#endif

    LSSP_PC_USER,               /* user defined */

} LSSP_PC_TYPE;
```

There are some internal preconditioners, including `NON` (no preconditioner), `ILUK` (point-wise ILU(k)), `ILUT` (point-wise ILUT(p, tol)), and `BILUK` (block-wise ILU(k)). However, `BILUK` requires `BLAS` and `LAPACK`. Users can set their own preconditioner if preconditioner type is set to LSSP_PC_USER. In this case, user should provide three functions, whose types are defined as LSSP_PC_ASSEMBLE, LSSP_PC_SOLVE and LSSP_PC_DESTROY.

LSSP also provides some external preconditioners, which are BILUT (block-wise ILUT(p, tol)), VBILUT (variable block-wise ILUT(p, tol)), VBILUK (variable block-wise ILU(k))and ARMS (multi-level method) from package ITSOL, AMG (algebraic multi-grid method) and FMG (full multi-grid method) from package FASP, and MI20AMG (MI20 AMG) from package HSL MI20.

4.3 Solver and Preconditioner Management

Figure 4.1 shows solution process, which includes the following steps:

1. setup matrix, unknown vector and right hand side vector, and initialize values;
2. create solver and preconditioner;
3. set solver and preconditioner parameters;
4. assemble solver and preconditioner;
5. solve the linear system;
6. get values and return to caller;
7. destroy solver, preconditioner, matrix and vectors;

4.3.1 Create

lssp_solver_create creates solver object and preconditioner object using solver type and preconditioner type.

```
void lssp_solver_create(LSSP_SOLVER &s, LSSP_SOLVER_TYPE s_type, LSSP_PC &pc,\
    LSSP_PC_TYPE p_type);
```

4.3.2 Assemble

lssp_solver_assemble assembles solver object and preconditioner object.

```
void lssp_solver_assemble(LSSP_SOLVER &s, lssp_mat_csr &Ax, lssp_vec x, \
    lssp_vec b, LSSP_PC &pc);
```

```
{
    lssp_mat_csr A;
    LSSP_SOLVER solver;
    LSSP_PC pc;
    lssp_vec x;
    lssp_vec b;

    /* 1: setup A, x, b */
    x = lssp_vec_create(n);
    b = lssp_vec_create(n);

    A = lssp_mat_create(int nrows, int ncols, int *Ap, int *Aj, double *Ax);
    for (i = 0; i < n; i++) {
        lssp_vec_set_value_by_index(x, i, double val);
        lssp_vec_set_value_by_index(b, i, double val);
    }

    /* 2: create solver and pc */
    lssp_solver_create(solver, LSSP_SOLVER_BICGSTAB, pc, LSSP_PC_ILUK);

    /* 3: settings */
    lssp_solver_set_restart(solver, m);
    lssp_solver_set_maxit(solver, itr_max);

    /* 4: assemble solver and preconditioner */
    lssp_solver_assemble(solver, A, x, b, pc);

    /* 5: solve */
    lssp_solver_solve(solver, pc);

    /* 6: get value */
    for (i = 0; i < n; i++) {
        value = lssp_vec_get_value_by_index(x, i);
    }

    /* 7: destroy pc and sover */
    lssp_solver_destroy(solver, pc);
    lssp_mat_destroy(A);
    lssp_vec_destroy(x);
    lssp_vec_destroy(b);
}
```

Figure 4.1: Solution process

4.3.3 Solve

`lssp_solver_solve` solves linear system.

```
int lssp_solver_solve(LSSP_SOLVER &solver, LSSP_PC &pc);
```

4.3.4 Destroy

`lssp_solver_destroy` destroys solver and preconditioner objects and releases internal memory.

```
void lssp_solver_destroy(LSSP_SOLVER &s, LSSP_PC &pc);
```

4.4 Solver Settings

4.4.1 General Settings

`lssp_solver_reset_rhs` rests right-hand side.

```
void lssp_solver_reset_rhs(LSSP_SOLVER &s, lssp_vec rhs);
```

`lssp_solver_reset_unknown` resets initial guess and solution vector.

```
void lssp_solver_reset_unknown(LSSP_SOLVER &s, lssp_vec x);
```

`lssp_solver_reset_type` resets solver type.

```
void lssp_solver_reset_type(LSSP_SOLVER &s, LSSP_SOLVER_TYPE type);
```

`lssp_solver_set_rtol` sets relative tolerence.

```
void lssp_solver_set_rtol(LSSP_SOLVER &s, double tol);
```

`lssp_solver_set_atol` sets absolute tolerence.

```
void lssp_solver_set_atol(LSSP_SOLVER &s, double tol);
```

`lssp_solver_set_rbtol` sets relative b norm tolerence.

```
void lssp_solver_set_rbtol(LSSP_SOLVER &s, double tol);
```

```
/* set maximal number of iteration */
void lssp_solver_set_maxit(LSSP_SOLVER &s, int maxit);
```

lssp_solver_set_restart set the number of restart.

```
void lssp_solver_set_restart(LSSP_SOLVER &s, int m);
```

lssp_solver_set_augk sets aug_k for LGMRES(m).

```
void lssp_solver_set_augk(LSSP_SOLVER &s, int k);
```

lssp_solver_set_bgsl set bgsl for BICGSTAB(l).

```
void lssp_solver_set_bgsl(LSSP_SOLVER &s, int k);
```

lssp_solver_set_idrs sets idrs for IDRS solver.

```
void lssp_solver_set_idrs(LSSP_SOLVER &s, int k);
```

lssp_solver_reset_verbosity resets verbosity of a solver.

```
void lssp_solver_reset_verbosity(LSSP_SOLVER &s, int v);
```

lssp_solver_get_residual gets residual.

```
double lssp_solver_get_residual(LSSP_SOLVER s);
```

lssp_solver_get_nits gets number of iteration.

```
int lssp_solver_get_nits(LSSP_SOLVER s);
```

lssp_solver_set_log sets log file handler.

```
void lssp_solver_set_log(LSSP_SOLVER &s, FILE *io);
```

4.4.2 AMG Solver Setting

lssp_solver_amg_reset_pars sets new parameters to AMG solver.

```
void lssp_solver_amg_reset_pars(LSSP_SOLVER &s, AMG_param par);
```

4.4.3 FASP Solver Setting

lssp_fasp_set_pars resets default parameters. If parameter pointer is not NULL, default parameters will be overriden.

```
void lssp_fasp_set_pars(LSSP_SOLVER *solver, input_param *inparam,
    itsolver_param *itsparam, AMG_param *amgparam, ILU_param *iluparam,
    Schwarz_param *schparam);
```

4.4.4 LIS Solver Setting

lssp_solver_lis_set_pars sets solver id and preconditioner id.

```
static const char * lis_solver[] = {
    "-i bicgstab",
    "-i bicgstabl",
    "-i cg",
    "-i cgs",
    "-i bicg",
    "-i bicgsafe",
    "-i bicr",
    "-i cr",
    "-i bicrstab",
    "-i bicrsafe",
    "-i idrs",
    "-i crs",
    "-i gpbicr",
    "-i gpbicg",
    "-i tfqmr",
    "-i orthomin",
    "-i gmres",
    "-i fgmres",
    "-i minres",
};

static const char * lis_pc[] = {
    "-p none",
    "-p ilut",
    "-p ilu -ilu_fill 1",
    "-p is",
    "-p sainv",
    "-p saamg -saamg_unsym -saamg_theta 0.5",
    "-p hybrid",
    "-p iluc",
    "-p ssor",
    "-p jacobi",
};

void lssp_solver_lis_set_pars(LSSP_SOLVER &solver, unsigned int solver_id,
```

```
    unsigned int pc_id);
```

4.4.5 SXAMG Setting

`lssp_solver_sxamg_set_pars` resets default parameters.

```
void lssp_solver_sxamg_set_pars(LSSP_SOLVER *solver, SX_AMG_PARS *pars);
```

4.4.6 PETSc Setting

`lssp_solver_petsc_setting` resets PETSc default parameters by a function.

```
typedef void (*solver_petsc_setting)(void *ksp, void *pc);

void lssp_solver_petsc_setting(solver_petsc_setting func);
```

4.5 Preconditioner Settings

4.5.1 ILUK Setting

`lssp_pc_iluk_set_level` sets ILUK level.

```
void lssp_pc_iluk_set_level(LSSP_PC &pc, int level);
```

4.5.2 ILUT Setting

`lssp_pc_ilut_set_drop_tol` and `lssp_pc_ilut_set_p` sets `tol` and `p` parameters of ILUT(tol, p), which overrides default parameters.

```
void lssp_pc_ilut_set_drop_tol(LSSP_PC &pc, double tol);
void lssp_pc_ilut_set_p(LSSP_PC &pc, int p);
```

4.5.3 AMG Setting

`lssp_pc_amg_set_pars` sets new parameters, which overrides default settings.

```
void lssp_pc_amg_set_pars(LSSP_PC &pc, AMG_param *amgparam);
```

4.5.4 SXAMG Setting

`lssp_pc_sxamg_set_pars` sets new parameters, which overrides default parameters.

```
void lssp_pc_sxamg_set_pars(LSSP_PC &pc, SX_AMG_PARS *pars);
```

Chapter 5

Utilities

5.1 Print

lssp_set_log sets log file handler.

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

lssp_printf outputs to stdout and default log file if set.

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

lssp_error prints output error message and quits with error code.

```
void lssp_error(int code, const char *fmt, ...);
```

lssp_warning print warning info.

```
void lssp_warning(const char *fmt, ...);
```

5.2 Memory

The following functions provide memory allocation, calloc, reallocation, freeing and copying.

```
template <typename T> T * lssp_malloc(const int n);

template <typename T> T * lssp_calloc(const int n);

template <typename T> T * lssp_realloc(T * old, const int n);

template <typename T> void lssp_free(T * &p);
```

```
template<typename T> void lssp_memcpy_on(T *dst, const T *src, const int n);

template <typename T> T * lssp_copy_on(const T *src, const int n);
```

5.3 Performance

`lssp_get_time` gets current time.

```
double lssp_get_time();
```

`lssp_get_mem_usage` returns current memory usage. If `peak` is not `NULL`, then peak memory is returned to `peak`.

```
double lssp_get_mem_usage(double *peak);
```