# SyLVER Documentation

***Release v2019-01-31***

**Florent Lopez**

**Mar 17, 2020**

# Contents:

1 Introduction 3
1.1 Purpose . . . . . . . . . . 3
1.2 Usage overview . . . . . . . . . . 4

2 Installation 5
2.1 Quick Start . . . . . . . . . . 5
2.2 Third-party libraries . . . . . . . . . . 5

3 API 9
3.1 General . . . . . . . . . . 9
3.2 SpLDLT . . . . . . . . . . 9

4 Data types 13
4.1 Options . . . . . . . . . . 13
4.2 Information . . . . . . . . . . 15

5 Examples 17
5.1 SpLDLT . . . . . . . . . . 17

6 Indices and tables 19

Index 21

SyLVER

CHAPTER 1

# Introduction

## 1.1 Purpose

SyLVER is a sparse direct solver for computing the solution of **large sparse symmetrically-structured linear systems** of equations. This includes both positive-definite and indefinite sparse symmetric systems as well as unsymmetric system whose sparsity pattern is symmetric.

The solution of the system of equations:

$$AX = B$$

is achived by computing a factorization of the input matrix. the following cases are covered:

1. $A$ is **symmetric positive-definite**, we compute the **sparse Cholesky factorization**:

$$PAP^T = LL^T$$

where the factor $L$ is a lower triangular matrix and the matrix $P$ is a permutation matrix used to reduce the fill-in generated during the factorization. Following the matrix factorization the solution can be retrieved by successively solving the system $LY = PB$ (forward substitution) and $L^T PX = Y$ (backward substitutions).

2. $A$ is **symmetric indefinite**, then we compute the sparse $LDL^T$ decomposition:

$$A = PLD(PL)^T$$

where $P$ is a permutation matrix, $L$ is unit lower triangular, and $D$ is block diagonal with blocks of size $1 \times 1$ and $2 \times 2$.

The code optionally supports hybrid computation using one or more NVIDIA GPUs.

SyLVER returns bit-compatible results.

An option exists to scale the matrix. In this case, the factorization of the scaled matrix $\overline{A} = SAS$ is computed, where $S$ is a diagonal scaling matrix.

## 1.2 Usage overview

Solving $AX = B$ using SyLVER is a four stage process.

- If $A$ is *symmetric*:
    1. Call `spldlt_analyse()` to perform a symbolic factorization, stored in *spldlt_akeep*.
    2. Call `spldlt_factor()` to perform a numeric factorization, stored in *spldlt_fkeep*. More than one numeric factorization can refer to the same *spldlt_akeep*.
    3. Call `spldlt__solve()` to perform a solve with the factors. More than one solve can be performed with the same *spldlt_fkeep*.
    4. Once all desired solutions have been performed, free memory with `spldlt_free()`.

**Note:** The `sylver_init()` routine must be called before any other routines whitin SyLVER. When all the desired operations have been performed, the `sylver_finalize()` routine should be called.

# CHAPTER 2

# Installation

## 2.1 Quick Start

Under Linux, or Mac OS X:

```
# Get latest development version from github
git clone https://github.com/NLAFET/sylver
cd sylver

mkdir build # create build directory
cd build
# configure compilation
cmake <path-to-source> -D SYLVER_RUNTIME=StarPU -D SYLVER_ENABLE_CUDA=ON
make # run compilation
```

## 2.2 Third-party libraries

### 2.2.1 SPRAL

SPRAL is an open-source library for sparse linear algebra and associated algorithm and has several important features used in SyLVER. By default, SPRAL is automatically download and built during the installation of SyLVER. However, if you wish to use your own version of SPRAL, which is not recommended, you can use the instructions below to install it.

#### SPRAL installation

The installation instruction presented here are only useful if you wish to install SPRAL as an internal package. By default SPRAL is automatically downloaded and built during the installation of SyLVER.

The latest release of SPRAL can be found on its GitHub repository. The compilation of SPRAL is handled by autotools and for example can be done as follow when using the GCC compilers:

```
cd spral
mkdir build
cd build
../configure CXX=g++ FC=gfortran CC=gcc CFLAGS="-g -O2 -march=native" CXXFLAGS="-g -
↪O2 -march=native" FCFLAGS="-g -O2 -march=native" --with-metis="-L/path/to/metis -
↪lmetis" --with-blas="-L/path/to/blas -lblas" --with-lapack="-L/path/to/lapack -
↪llapack" --disable-openmp --disable-gpu
make
```

Note that the compilation flags used for SPRAL **must match** the flags used in the compilation of SyLVER. Here we use the flags `-g -O2 -march=native` that correspond to the `RelWithDebInfo` build type in SyLVER.

Here we use the `--disable-openmp` option because SyLVER works with the serial version of SPRAL. Additionally, in this example we disabled the compilation of the SPRAL GPU kernels using the `--disable-gpu` option.

**Sequential version** of BLAS and LAPACK should be used. We recommend using the MKL library for best performance on Intel machines and ESSL on IBM machines. The MKL link line advisor can be useful to fill the `--with-blas` and `--with-lapack` options.

When compiling SyLVER you need to provide both the path to the SPRAL source directory which can be given using the `-D SPRAL_SRC_DIR` CMake option or the `SPRAL_SRC_DIR` environment variable and the path to the SPRAL library which can be given using the `-D SPRAL_DIR` CMake option or the `SPRAL_DIR` environment variable.

### 2.2.2 METIS

The MeTiS partitioning library is needed by the SPRAL library and therefore, needed when linking the SyLVER package for building examples and test drivers.

When compiling SyLVER you can provide the path to the MeTiS library using either `-D METIS_DIR` CMake option or the `METIS_DIR` environment variable.

### 2.2.3 hwloc

The hwloc library is topology discovery library which is necessary for linking the examples and test drivers if SPRAL was compiled with it. In this case, the library path can be given to CMake using either the `-D HWLOC_DIR` definition or the `HWLOC_DIR` environment variable.

### 2.2.4 Runtime system

The `-D SYLVER_RUNTIME=StarPU` enables the compilation of the parallel version of SyLVER using StarPU runtime system. In this case the StarPU version needs to be at least 1.3.0. The StarPU library is found with the `FindSTARPU.cmake` script located in the `cmake/Modules` directory. Note that, for this script to be able to find the StarPU library, you need to set the environment variable `STARPU_DIR` to the path of you StarPU install base directory.

### 2.2.5 BLAS and LAPACK

The BLAS and LAPACK libraries play an important role in the performance of the solver. We recommend using the MKL library for best performance on Intel machines and the ESSL library when running on IBM machines. Alternative BLAS and LAPACK libraries include OpenBLAS. Note that SyLVER should be linked against the **sequential** BLAS and LAPACK libraries.

These libraries are found via the CMake scripts FindBLAS and FindLAPACK and therefore it is possible to use the options `-D BLA_VENDOR` to indicate which libraries to use. For example:

```
cmake <path-to-source> -D BLA_VENDOR=Intel10_64lp_seq # configure compilation
```

selects and locates the sequential BLAS and LAPACK implementation for the compilation and when linking test drivers, example and tests.

If CMake is unable to locate the requested libraries via the `-D BLA_VENDOR`, it is still possible to give them explicitly using the `-D LBLAS` and `-D LLAPACK` options. For example:

```
# configure compilation
cmake <path-to-source> -D LBLAS="-L/path/to/blas -lblas" -D LLAPACK="-L/path/to/
↪lapack -llapack"
```

CHAPTER 3

# API

In the below, all reals are double precision unless otherwise indicated.

## 3.1 General

**`subroutine sylver_init`**(*ncpu*, *ngpu*)

Initialization routine which should be called before any other routine within SyLVER. The number of CPUs and GPUs involved in the computations should be passed to this routine.

**Parameters**

- **ncpu** *[integer,in]* :: number of CPUs to be used in the execution of SyLVER routines.
- **ngpu** *[integer,in]* :: number of GPUs to be used in the execution of SyLVER routines. Note that if CUDA is not enabled during the compilation, this value will be ignored.

**`subroutine sylver_finalize`**()

SyLVER termination routine which should be called once all the desired operations have been performed.

## 3.2 SpLDLT

**`subroutine spldlt_analyse`**(*akeep*, *n*, *ptr*, *row*, *options*, *inform*[, *order*, *val*, *ncpu*, *ngpu*, *check* ])

Perform the analyse (symbolic) phase of the factorization for a matrix supplied in Compressed Sparse Column (CSC) format. The resulting symbolic factors stored in `spldlt_akeep` should be passed unaltered in the subsequent calls to ssids_factor().

**Parameters**

- **akeep** *[spldlt_akeep,out]* :: returns symbolic factorization, to be passed unchanged to subsequent routines.
- **n** *[integer,in]* :: number of columns in $A$.
- **ptr** (n+1) *[long,in]* :: column pointers for $A$ (see CSC format).

- **row** (ptr(n+1)-1) *[integer,in]* :: row indices for $A$ (see CSC format).
- **options** *[sylver_options,in]* :: specifies algorithm options to be used (see `sylver_options`).
- **inform** *[sylver_inform,out]* :: returns information about the execution of the routine (see `sylver_inform`).

**Options**

- **order** (n) *[integer,inout]* :: on entry a user-supplied ordering (options%ordering=0). On return, the actual ordering used (if present).
- **val** (ptr(n+1)-1) *[real,in]* :: non-zero values for $A$ (see CSC format). Only used if a matching-based ordering is requested.
- **ncpu** *[integer,in]* :: Number of CPU available for the execution. If absent, the value of ncpu passed to the `sylver_init()` routine is used instead.
- **ncpu** :: Number of GPU available for the execution. This value is ignored if CUDA is not enabled during the compilation and if absent, the value of ncpu passed to the `sylver_init()` routine is used instead.
- **check** *[logical,in]* :: if true, matrix data is checked. Out-of-range entries are dropped and duplicate entries are summed.

**Note:** If a user-supplied ordering is used, it may be altered by this routine, with the altered version returned in order(:). This version will be equivalent to the original ordering, except that some supernodes may have been amalgamated, a topological ordering may have been applied to the assembly tree and the order of columns within a supernode may have been adjusted to improve cache locality.

**subroutine spldlt_factorize** (*akeep*, *fkeep*, *posdef*, *val*, *options*, *inform*[, *scale*, *ptr*, *row* ])

**Parameters**

- **akeep** *[spldlt_akeep,out]* :: symbolic factorization returned by preceding call to `spldlt_analyse()`.
- **akeep** :: returns numeric factorization, to be passed unchanged to subsequent routines.
- **posdef** *[logical,in]* :: true if matrix is positive-definite.
- **val** (*) *[real,in]* :: non-zero values for $A$ in same format as for the call to `spldlt_analyse()`.
- **options** *[sylver_options,in]* :: specifies algorithm options to be used (see `sylver_options`).
- **inform** *[sylver_inform,out]* :: returns information about the execution of the routine (see `sylver_inform`).

**Options**

- **scale** (n) *[real,inout]* :: diagonal scaling. scale(i) contains entry $S_{ii}$ of $S$. Must be supplied by user if `options%scaling=0` (user-supplied scaling). On exit, return scaling used.
- **ptr** (n+1) *[integer(long),in]* :: column pointers for $A$, only required if scaling is required (options%scaling > 0) expect in the case where matching-based ordering is done (options%scaling = 3)

- **row** (ptr(n+1)-1) *[integer,in]* :: row indices for $A$, only required if scaling is required (options%scaling > 0) expect in the case where matching-based ordering is done (options%scaling = 3)

**subroutine spldlt_solve** (*akeep*, *fkeep*, *nrhs*, *x*, *ldx*, *options*, *inform*[, *job* ])

Solve (for multiple right-hand sides) one of the following equations:

| *job* | Equation solved |
|---|---|
| 0 (or absent) | $AX = B$ |
| 1 | $PLX = SB$ |
| 2 | $DX = B$ |
| 3 | $(PL)^T S^{-1} X = B$ |
| 4 | $D(PL)^T S^{-1} X = B$ |

Recall $A$ has been factorized as either:

- $SAS = (PL)(PL)^T$ (positive-definite case); or
- $SAS = (PL)D(PL)^T$ (indefinite case).

**Parameters**

- **akeep** *[spldlt_akeep,in]* :: symbolic factorization returned by preceding call to `spldlt_analyse()`
- **fkeep** *[spldlt_fkeep,in]* :: numeric factorization returned by preceding call to `spldlt_factor()`.
- **nrhs** *[integer,in]* :: number of right-hand sides.
- **x** (ldx,nrhs) *[real,inout]* :: right-hand sides $B$ on entry, solutions $X$ on exit.
- **ldx** *[integer,in]* :: leading dimension of `x`.
- **options** *[sylver_options,in]* :: specifies algorithm options to be used (see `sylver_options`).
- **inform** *[sylver_inform,out]* :: returns information about the execution of the routine (see `sylver_inform`).

**Options** **job** *[integer,in]* :: specifies equation to solve, as per above table.

# CHAPTER 4

# Data types

## 4.1 Options

**type sylver_options**

The derived data type *sylver_options* is used to specify the options used within SyLVER. The components, that are automatically given default values in the definition of the type, are:

**Type fields**

- **% print_level** *[integer,default=0]* :: the level of printing. The different levels are:

| | |
|---|---|
| $< 0$ | No printing. |
| $= 0$ | Error and warning messages only. |
| $= 1$ | As 0, plus basic diagnostic printing. |
| $> 1$ | As 1, plus some additional diagnostic printing. |

- **% unit_diagnostics** *[integer,default=6]* :: Fortran unit number for diagnostics printing. Printing is suppressed if <0.
- **% unit_error** *[integer,default=6]* :: Fortran unit number for printing of error messages. Printing is suppressed if <0.
- **% unit_warning** *[integer,default=6]* :: Fortran unit number for printing of warning messages. Printing is suppressed if <0.
- **% ordering** *[integer,default=1]* :: Ordering method to use in analyse phase:

| | |
|---|---|
| 0 | User-supplied ordering is used (*order* argument to `spldlt_analyse()` or `splu_analyse()`). |
| 1 (default) | METIS ordering with default settings. |
| 2 | Matching-based elimination ordering is computed (the Hungarian algorithm is used to identify large off-diagonal entries. A restricted METIS ordering is then used that forces these on to the subdiagonal).<br>**Note:** This option should only be chosen for indefinite systems. A scaling is also computed that may be used in `spldlt_factor()` or `splu_factor()` (see %scaling below). |

- `% nemin` *[integer,default=32]* :: supernode amalgamation threshold. Two neighbours in the elimination tree are merged if they both involve fewer than nemin eliminations. The default is used if nemin<1.
- `% use_gpu` *[logical,default=true]* :: Use an NVIDIA GPU if present.
- `% scaling` *[integer,default=0]* :: scaling algorithm to use:

| | |
|---|---|
| <=0 (default) | No scaling (if `scale(:)` is not present on call to `spldlt_factor()` or `splu_factor()`, or user-supplied scaling (if `scale(:)` is present). |
| =1 | Compute using weighted bipartite matching via the Hungarian Algorithm (`MC64` algorithm). |
| =2 | Compute using a weighted bipartite matching via the Auction Algorithm (may be lower quality than that computed using the Hungarian Algorithm, but can be considerably faster). |
| =3 | Use matching-based ordering generated during the analyse phase using options%ordering=2. The scaling will be the same as that generated with options%scaling= 1 if the matrix values have not changed. This option will generate an error if a matching-based ordering was not used during analysis. |
| >=4 | Compute using the norm-equilibration algorithm of Ruiz. |

- `% nb` *[integer,default=256]* :: Block size to use for parallelization of large nodes on CPU resources.
- `% pivot_method` *[integer,default=1]* :: Pivot method to be used on CPU, one of:

| | |
|---|---|
| 0 | Aggressive a posteori pivoting. Cholesky-like communication pattern is used, but a single failed pivot requires restart of node factorization and potential recalculation of all uneliminated entries. |
| 1 (default) | Block a posteori pivoting. A failed pivot only requires recalculation of entries within its own block column. |
| 2 | Threshold partial pivoting. Not parallel. |

- `% small` *[real,default=1d-20]* :: threshold below which an entry is treated as equivalent to *0.0*.
- `% u` *[real,default=0.01]* :: relative pivot threshold used in symmetric indefinite case. Values outside of the range $[0, 0.5]$ are treated as the closest value in that range.

## 4.2 Information

**type sylver_inform**

The derived data type *sylver_inform* is used to return information about the progress and needs of the algorithm that might be of interest for the user.

**Type fields**

- `%` **flag** *[integer]* :: exit status of the algorithm (see table below).
- `%` **cublas_error** *[integer]* :: CUBLAS error code in the event of a CUBLAS error (0 otherwise).
- `%` **cuda_error** *[integer]* :: CUDA error code in the event of a CUDA error (0 otherwise). Note that due to asynchronous execution, CUDA errors may not be reported by the call that caused them.
- `%` **matrix_dup** *[integer]* :: number of duplicate entries encountered (if *spldlt_analyse()* or `splu_analyse()` called with check=true).
- `%` **matrix_missing_diag** *[integer]* :: number of diagonal entries without an explicit value (if *spldlt_analyse()* or `splu_analyse()` called with check=true).
- `%` **matrix_outrange** *[integer]* :: number of out-of-range entries encountered (if *spldlt_analyse()* or `splu_analyse()` called with check=true).
- `%` **matrix_rank** *[integer]* :: (estimated) rank (structural after analyse phase, numerical after factorize phase).
- `%` **maxdepth** *[integer]* :: maximum depth of the assembly tree.
- `%` **maxfront** *[integer]* :: maximum front size (without pivoting after analyse phase, with pivoting after factorize phase).
- `%` **num_delay** *[integer]* :: number of delayed pivots. That is, the total number of fully-summed variables that were passed to the father node because of stability considerations. If a variable is passed further up the tree, it will be counted again.
- `%` **num_factor** *[long]* :: number of entries in $L$ (without pivoting after analyse phase, with pivoting after factorize phase).
- `%` **num_flops** *[long]* :: number of floating-point operations for Cholesky factorization (indefinte needs slightly more). Without pivoting after analyse phase, with pivoting after factorize phase.
- `%` **num_neg** *[integer]* :: number of negative eigenvalues of the matrix $D$ after factorize phase.
- `%` **num_sup** *[integer]* :: number of supernodes in assembly tree.
- `%` **num_two** *[integer]* :: number of $2 \times 2$ pivots used by the factorization (i.e. in the matrix $D$ in the indefinite

  case).
- `%` **stat** *[integer]* :: Fortran allocation status parameter in event of allocation error (0 otherwise).

| inform%flag | Return status |
|---|---|
| 0 | Success. |
| -1 | Error in sequence of calls (may be caused by failure of a preceding call). |
| -2 | n<0 or ne<1. |
| -3 | Error in ptr(:). |
| -4 | CSC format: All variable indices in one or more columns are out-of-range.<br>Coordinate format: All entries are out-of-range. |
| -5 | Matrix is singular and options%action=.false. |
| -6 | Matrix found not to be positive definite but posdef=true. |
| -7 | ptr(:) and/or row(:) not present although required. |
| -8 | options%ordering out of range, or options%ordering=0 and order parameter not provided or not a valid permutation. |
| -9 | options%ordering=-2 but val(:) was not supplied. |
| -10 | ldx<n or nrhs<1. |
| -11 | job is out-of-range. |
| -13 | Called `spldlt_enquire_posdef()` on indefinite factorization. |
| -14 | Called `spldlt_enquire_indef()` on positive-definite factorization. |
| -15 | options%scaling=3 but a matching-based ordering was not performed during analyse phase. |
| -50 | Allocation error. If available, the stat parameter is returned in inform%stat. |
| -51 | CUDA error. The CUDA error return value is returned in inform%cuda_error. |
| -52 | CUBLAS error. The CUBLAS error return value is returned in inform%cublas_error. |
| +1 | Out-of-range variable indices found and ignored in input data. inform%matrix_outrange is set to the number of such entries. |
| +2 | Duplicate entries found and summed in input data. inform%matrix_dup is set to the number of such entries. |
| +3 | Combination of +1 and +2. |
| +4 | One or more diagonal entries of $A$ are missing. |
| +5 | Combination of +4 and +1 or +2. |
| +6 | Matrix is found be (structurally) singular during analyse phase. This will overwrite any of the above warning flags. |
| +7 | Matrix is found to be singular during factorize phase. |
| +8 | Matching-based scaling found as side-effect of matching-based ordering ignored (consider setting options%scaling=3). |

CHAPTER 5

# Examples

## 5.1 SpLDLT

Suppose we wish to factorize the matrix

$$A = \begin{pmatrix} 2. & 1. & & & \\ 1. & 4. & 1. & & 1. \\ & 1. & 3. & 2. & \\ & & 2. & -1. & \\ & 1. & & & 2. \end{pmatrix}$$

and then solve for the right-hand side

$$B = \begin{pmatrix} 4. \\ 17. \\ 19. \\ 2. \\ 12. \end{pmatrix}.$$

The following code may be used.

```
program spldlt_example
  use sylver_mod
  implicit none

  ! Derived types
  type (spldlt_akeep_type)   :: akeep
  type (spldlt_fkeep_type)   :: fkeep
  type (sylver_options) :: options
  type (sylver_inform)  :: inform

  ! Parameters
  !integer, parameter :: long = selected_int_kind(16)
  !integer, parameter :: wp = kind(0.0d0)
  ! Matrix data
  logical :: posdef
  integer :: n, row(9)
  integer(long) :: ptr(6)
  real(wp) :: val(9)

  ! Other variables
  integer :: ncpu, ngpu
  integer :: nrhs
  real(wp) :: x(5)

  ! Data for matrix:
  ! ( 2  1          )
  ! ( 1  4  1     1 )
  ! (    1  3  2    )
  ! (       2 -1    )
  ! (    1        2 )
  posdef = .false.
  n = 5
  ptr(1:n+1)        = (/ 1,        3,              6,        8,    9,  10 /)
  row(1:ptr(n+1)-1) = (/ 1,   2,   2,   3,   5,   3,   4,   4,    5   /)
  val(1:ptr(n+1)-1) = (/ 2.0, 1.0, 4.0, 1.0, 1.0, 3.0, 2.0, -1.0, 2.0 /)

  ! The right-hand side with solution (1.0, 2.0, 3.0, 4.0, 5.0)
  nrhs = 1
  x(1:n) = (/ 4.0, 17.0, 19.0, 2.0, 12.0 /)

  ncpu = 8
  ngpu = 0

  call sylver_init(ncpu, ngpu)

  ! Perform analyse and factorize
  call spldlt_analyse(akeep, n, ptr, row, options, inform, ncpu=ncpu, ngpu=ngpu)
  if(inform%flag<0) go to 100

  call spldlt_factorize(akeep, fkeep, posdef, val, options, inform)
  if(inform%flag<0) go to 100

  call spldlt_solve(akeep, fkeep, nrhs, x, n, options, inform)
  if(inform%flag<0) go to 100
  write(*,'(a,/,(3es18.10))') ' The computed solution is:', x(1:n)

  call sylver_finalize()

100 continue
  call spldlt_akeep_free(akeep)
  call spldlt_fkeep_free(fkeep)

end program spldlt_example
```

This produces the following output:

```
The computed solution is:
 1.0000000000E+00  2.0000000000E+00  3.0000000000E+00
 4.0000000000E+00  5.0000000000E+00
Pivot order:   -3    4   -1    0   -2
```

# CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

# Index

## S

spldlt_analyse() (fortran subroutine), **9**
spldlt_factorize() (fortran subroutine), **10**
spldlt_solve() (fortran subroutine), **11**
sylver_finalize() (fortran subroutine), **9**
sylver_inform (fortran type), **15**
sylver_init() (fortran subroutine), **9**
sylver_options (fortran type), **13**