Numerical Libraries with C or Fortran

Shaohao Chen
Research Computing, IS&T, Boston University
Outline

2. Fast Fourier transform: FFTw
3. Linear algebra libraries: LAPACK/BLAS
4. Intel Math Kernel Library (MKL)
5. Krylov subspace solver: PETSc
6. GNU scientific libraries (GSL)
1. Overview

What you will learn today

• Basic knowledge of numerical libraries.
• How to check available libraries on BU SCC.
• How to use numerical libraries on BU SCC.
• Basic programming with several numerical libraries: FFTw, LAPACK/BLAS, MKL, PETSc, GSL
What is numerical library?

• What is the definition of a library in computer science?
  In computer science, a library is a collection of non-volatile resources used by computer programs, often to develop software. These may include configuration data, documentation, help data, message templates, pre-written code and subroutines, classes, values or type specifications. (from wiki)

• What is numerical library?
  Numerical library is collection of functions, subroutines or classes that implement mathematical or numerical methods for a certain subject. Usually these functions or routines are common and can be used to build computer programs for various research fields.
Several widely-used numerical libraries

• Fastest Fourier Transform in the West (FFTW) computes Fourier and related transforms. Written in C. Fortran interface is available.

• Basic Linear Algebra Subprograms (BLAS) performs basic vector and matrix operations. Linear Algebra Package (LAPACK) provides linear algebra routines based on BLAS. Written in Fortran. C interface (CBLAS/LAPACKE) is available.

• Intel Math Kernel Library (MKL) includes optimized LAPACK, BLAS, FFT, Vector Math and Statistics functions. C/C++ and Fortran interfaces are available.

• Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. Written in C++. Fortran interface is available.

• GNU Scientific Library (GSL) provides a wide range of mathematical routines. Written in C++. Fortran interface (FGSL) is under development.

• For more: http://en.wikipedia.org/wiki/List_of_numerical_libraries
Why numerical libraries?

• Many functions or subroutines you need may have already been coded by others. Not necessary to code every line by yourself.
• Do not reinvent the wheel. Always Check available libraries before start writing your program.
• Save your time and efforts!

Advantages of using numerical libraries:
• Computing optimizations
• Parallelization
• Portability
• Easy to debug, easy to read
Prerequisites

Compilers: to compile source codes
• **Intel**: icc, icpc, ifort
• **GNU**: gcc, g++, gfortran
• **PGI**: pgcc, pgc++, pgf90

MPI implementations: to enable parallel computation with MPI standard
• mpich
• mvapich2
• openmpi
• impi
Install numerical libs

A typical three-step installation:

• **configure**: configure machine-dependent environments
• **make**: compiles source codes based on settings in the makefile
• **make install**: copy installed libs and binaries to a destination

Other types of installation:

• **manually modify Makefile**, then make
• **cmake**: machine-dependent make
Available libraries on BU SCC

• Check BU Research Computing software webpage:
  http://sccsvc.bu.edu/software/#/
  under the libraries catalog.

• Use module to check libraries on SCC:
  module av
  module whatis
  module list
  module show
How to use numerical libs

Step 1: Modify (a little) source code:
• Call functions or subroutines provided by the libs.
• Include necessary head files.

Step 2: Compile and link (see next page):
• Set paths to lib and include directories for numerical libs (use module or set manually).
• Compile your own code and link it to precompiled numerical libs.
• The same compilers should be used for numerical libs and for your own codes.

Step 3: Run the program:
• Set LD_LIBARRY_PATH, if runtime(dynamic) libraries are used.
Compile and link

• Execute `module show software_name` to get the paths to header files and lib files.

• Compile your own source code:
  
  ```
  ${compiler} -c -I/path/to/include name.c (or name.f)
  ```

• Link to libs and build the binary
  
  ➢ Use a lib installed at a specific location (such as /share/pkg on SCC)
    
    ```
    ${compiler} name.o -L/path/to/lib -l${libname} -o name
    ```
  
  ➢ Force to use a static lib
    
    ```
    ${compiler} name.o -L/path/to/lib -static -l${libname} -o name
    ```
Static libs

• A static lib is typically named as libname.a
• A static lib (*.a file) is an archive of a bunch of object (*.o) files.
• A program using a static library extracts the code that it uses from the static library and makes it part of the program.

Advantages compared to shared libs:
• There is no additional run-time loading costs.
• Once built, the final binary has no dependencies on the library.

Disadvantages compared to shared libs:
• Larger size of binary, larger launch time, larger memory usage at run time.
• For any change(up-gradation) in the library, every time you have to recompile all programs that use it.
Shared(Dynamic) libs

• Shared libs are typically named as libname.so or libname.so.*.
• A program using a shared library only makes reference to the code that it uses in the shared library.

Advantages compared to static libs:
• Smaller size of binary, less launch time, less memory usage at run time.
• If there is a change (up-gradation) in the library, you may not need to recompile the main programs.

Disadvantages compared to static libs:
• There is additional run-time loading costs.
• The final binary depends on the library at run time.
Additional settings to use shared libs

• To use static libs, set up environmental valuables for run-time access
  For bash: `export LD_LIBRARY_PATH=/path/to/lib`
  Alternatively, use module: `module load software_name`
  For csh/tcsh: `setenv LD_LIBRARY_PATH /path/to/lib`

Notes:
• The binary can “see” the dynamic libs under `${LD_LIBRARY_PATH}`.
• Especially for a parallel job that runs on multi nodes, LD_LIBRARY_PATH should be set for every node. Set it in the batch script.
2. Fast Fourier Transform in the west: FFTw

Main features:

• Library for computing the discrete Fourier transform (DFT)
• One or more dimensions FFT
• Arbitrary input size
• Both real and complex data
• Even/odd data, i.e. the discrete cosine/sine transforms
• Efficient handling of multiple, strided transforms
• Parallel transforms: parallelized with some flavor of threads (e.g. POSIX) or OpenMP. MPI version available in FFTW 3.3.
FFTw basics

Data type
- `fftw_complex`
- `fftw_plan`

Allocate and deallocate data
- `fftw_malloc`
- `fftw_free`

FFT plan and execution
- FFT plan functions (see next pages)
- `fftw_execute` // execute FFT plan
- `fftw_destroy_plan`
FFTw plan functions I

- **Complex DFT:**
  \[ X_k \stackrel{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z} \]

- **Inverse Complex DFT:**
  \[ x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}, \quad n \in \mathbb{Z} \]

**One dimensional**
- `fftw_plan_dft_1d(int n, fftw_complex *in, fftw_complex *out, int sign, unsigned flags);`
  - `sign`: either `FFTW_FORWARD` (-1) or `FFTW_BACKWARD` (+1).
  - `flags`: either `FFTW_MEASURE` or `FFTW_ESTIMATE`

**Multi dimensional**
- `fftw_plan_dft_2d` // two dimensions
- `fftw_plan_dft_3d` // three dimensions
- `fftw_plan_dft` // arbitrary dimensions
FFTw plan functions II

Real DFTs

- `fftw_plan_r2r_1d(int n, double *in, double *out, fftw_r2r_kind kind, unsigned flags)`
  kind: `FFTW_REDFT00`, `FFTW_RODFT00`, etc. For different types of even or odd transforms.
- `fftw_plan_r2r_2d`, `fftw_plan_r2r_3d`, `fftw_plan_r2r`

Real input, complex output, always FFTW_FORWARD

- `fftw_plan_dft_r2c_1d`, `fftw_plan_dft_r2c_2d`
- `fftw_plan_dft_r2c_3d`, `fftw_plan_dft_r2c`

Complex input, real output, always FFTW_BACKWARD

- `fftw_plan_dft_c2r_1d`, `fftw_plan_dft_c2r_2d`
- `fftw_plan_dft_c2r_3d`, `fftw_plan_dft_c2r`
Exercise 1: Fourier transform with FFTw

- **Task**: Compute the Fourier transform of a one-dimensional complex array, and compute the inverse Fourier transform of the output, which should be the same as the original input data.
Solution for Exercise 1 in C

- Source code: /project/scv/examples/numlibs/fftw/fftw3_prb.c
  - Include fftw head file: # include <fftw3.h>
  - Call fftw functions: fftw_malloc, fftw_plan_dft_1d, fftw_execute, etc.

- Compile and run

  module load fftw/3.3.4      # load fftw by module
  module show fftw/3.3.4     # show fftw-related environments
  gcc -c fftw3_prb.c -I/share/pkg/fftw/3.3.4/install/include   # compile
  gcc fftw3_prb.o -L/share/pkg/fftw/3.3.4/install/lib -lfftw3 -o fftw3_prb   # link
  ldd ./fftw3_prb      # check whether the binary is linked to fftw runtime libs
  ./fftw3_prb       # run
3. Linear algebra libraries

History:

- **LINPACK (LINear algebra PACKage):** since 1974
  - based on level-1 BLAS
- **LAPACK (Linear Algebra PACKage):** since 1989
  - based on level-3 BLAS, vectorized and threaded in Intel MKL
- **ScaLAPACK (Scalable LAPACK):** since 1995
  - parallel with MPI, for distributed memory systems, only a subset of LAPACK routines
- **DPLASMA (Distributed Parallel Linear Algebra Software for Multicore Architectures):** 2000’s
  - parallel for shared memory systems
- **MAGMA (Matrix Algebra for GPUs and Multicore Architectures):** 2000’s
  - parallel for GPU
- **Matlab:** a commercial software developed from LINPACK.
• Provides routines for performing basic vector and matrix operations.
• Level 1 BLAS: scalar, vector and vector-vector operations
• Level 2 BLAS: matrix-vector operations
• Level 3 BLAS: matrix-matrix operations

• Contents of compute routines:
  ➢ Matrix-matrix, matrix-vector addition and multiplication, etc.

Refer to user guide at http://www.netlib.org/blas/#_documentation
LAPACK

• Provides routines for solving systems of linear equations, linear least-squares problems, eigenvalue problems, and matrix factorizations.
• Written in Fortran 90.
• Can be seen as the successor to the linear equations and linear least-squares routines of LINPACK and the eigenvalue routines of EISPACK.

• Contents of compute routines:
  ➢ Linear Equations
  ➢ Generalized Orthogonal Factorizations
  ➢ Singular Value Decomposition
  ➢ Linear Least Squares Problems
  ➢ Symmetric and Nonsymmetric Eigen Problems

Refer to user guide at http://www.netlib.org/lapack/lug/node37.html
Exercise 2: Matrix product with LAPACK/BLAS

Task: Compute the real matrix product $C=\alpha A^*B+\beta C$ using LAPACK subroutine DGEMM, where $A$, $B$, and $C$ are matrices and $\alpha$ and $\beta$ are double precision scalars.
Solution for Exercise 2 in Fortran

- Source code: `/project/scv/examples/numlibs/lapack/matprod.f`
  - Initialize data for matrices A, B, C and real scalars alpha, beta.
  - Call LAPACK function: `DGEMM`

- Compile and run
  ```
  module show lapack/3.6.0  # show lapack-related environments
  gfortran matprod.f -L/share/pkg/lapack/3.6.0/install/lib -llapack -lblas -o matprod  # compile and link
  ./matprod  # run
  ```
4. Intel MKL

- Optimization for intel processors.
- Accelerates math processing routines that increase application performance and reduce development time.
- Includes highly vectorized and threaded Lapack, FFT, Vector Math and Statistics functions.
- Xeon-phi enabled.
## MKL LAPACK subroutines I

<table>
<thead>
<tr>
<th>Routine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?geev</td>
<td>Computes the eigenvalues and, optionally, the left and/or right eigenvectors of a general matrix.</td>
</tr>
<tr>
<td>?gels</td>
<td>Uses QR or LQ factorization to solve an overdetermined or underdetermined linear system with a full rank matrix.</td>
</tr>
<tr>
<td>?gelsd</td>
<td>Computes the minimum norm solution to a linear least squares problem using the singular value decomposition of $A$ and a divide and conquer method.</td>
</tr>
<tr>
<td>?gesdd</td>
<td>Computes the singular value decomposition of a general rectangular matrix using a divide and conquer algorithm.</td>
</tr>
<tr>
<td>?gesv</td>
<td>Computes the solution to the system of linear equations with a square matrix $A$ and multiple right-hand sides.</td>
</tr>
<tr>
<td>?gesvd</td>
<td>Computes the singular value decomposition of a general rectangular matrix.</td>
</tr>
<tr>
<td>?heev</td>
<td>Computes all the eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.</td>
</tr>
<tr>
<td>?heevd</td>
<td>Computes all the eigenvalues and, optionally, all the eigenvectors of a complex Hermitian matrix using a divide and conquer algorithm.</td>
</tr>
</tbody>
</table>

MKL LAPACK subroutines II

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heevr</code></td>
<td>Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix using the Relatively Robust Representations.</td>
</tr>
<tr>
<td><code>heevx</code></td>
<td>Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.</td>
</tr>
<tr>
<td><code>hesv</code></td>
<td>Computes the solution to the system of linear equations with a Hermitian matrix $A$ and multiple right-hand sides.</td>
</tr>
<tr>
<td><code>posv</code></td>
<td>Computes the solution to the system of linear equations with a symmetric or Hermitian positive definite matrix $\overline{A}$ and multiple right-hand sides.</td>
</tr>
<tr>
<td><code>syev</code></td>
<td>Computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.</td>
</tr>
<tr>
<td><code>syevd</code></td>
<td>Computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric matrix using a divide and conquer algorithm.</td>
</tr>
<tr>
<td><code>syevr</code></td>
<td>Computes the selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix using the Relatively Robust Representations.</td>
</tr>
<tr>
<td><code>syevx</code></td>
<td>Computes the selected eigenvalues and, optionally, the eigenvectors of a symmetric matrix.</td>
</tr>
<tr>
<td><code>sysv</code></td>
<td>Computes the solution to the system of linear equations with a real or complex symmetric matrix $A$ and multiple right-hand sides.</td>
</tr>
</tbody>
</table>
Exercise 3: Solve a linear system with MKL subroutines

- **Task:** Compute the solution to the system of linear equations $AX=B$ with a square matrix $A$ and multiple right-hand sides $B$ using the MKL routine `dgesv`. 
Solution for Exercise 3 in C

- **Source code:** `/project/scv/examples/numlibs/mkl/dgesv_ex.c`
  - Initialize data for matrices A and B
  - Call the MKL LAPACK function: `dgesv`

- **Compile and run**
  
  module load intel/2016
  icc -mkl dgesv_ex.c -o dgesv_ex
  ./dgesv
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, shared memory pthreads, and GPUs through CUDA or OpenCL, as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism.

Efficient for sparse-matrix problems
Parallel Numerical Components of PETSc

<table>
<thead>
<tr>
<th>Nonlinear Solvers</th>
<th>Time Steppers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newton-based Methods</td>
<td>Euler</td>
</tr>
<tr>
<td>Line Search</td>
<td>Backward Euler</td>
</tr>
<tr>
<td>Trust Region</td>
<td>Pseudo Time Stepping</td>
</tr>
<tr>
<td>Other</td>
<td>Other</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Krylov Subspace Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>GMRES</td>
</tr>
<tr>
<td>Richardson</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Preconditioners</th>
</tr>
</thead>
<tbody>
<tr>
<td>Additive Schwarz</td>
</tr>
<tr>
<td>LU (Sequential only)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Matrices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compressed Sparse Row (AIJ)</td>
</tr>
<tr>
<td>Block Diagonal (BDIAG)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices</td>
</tr>
</tbody>
</table>
PETSc Basics I

• PetscInitialize \// call MPI_Init
• PetscFinalize \// call MPI_Finalize

• Data types:
  PetscInt, PetscScalar, Vec, Mat

• Create objects:
  VecCreate(MPI_Comm comm, Vec *vec)
  MatCreate(MPI_Comm comm, Mat *mat)

• Destroy objects
  VecDestroy(Vec *vec)
  MatDestroy(Mat *mat)
PETSc Basics II

• Set sizes of objects
  
  \texttt{VecSetSizes} (\texttt{Vec v, PetscInt n, PetscInt N}) \quad // \text{ local size } n, \text{ global size } N
  
  \texttt{MatSetSizes} (\texttt{Mat A, PetscInt m, PetscInt n, PetscInt M, PetscInt N}) \quad // \text{ local size } m, n, \text{ global size } M, N

• Set values of objects
  
  \texttt{VecSetValues} (\texttt{Vec x, PetscInt ni, const PetscInt ix[], const PetscScalar y[], InsertMode mode})
  
  \texttt{MatSetValues} (\texttt{Mat mat, PetscInt m, const PetscInt idxm[], PetscInt n, const PetscInt idxn[], const PetscScalar v[], InsertMode mode}) \quad // \text{ Set values of a block. Unset blocks are filled with zero.}

  \quad \text{mode: either} \ \texttt{INSERT\_VALUES} \ \text{or} \ \texttt{ADD\_VALUES}
• Assembly
  VecAssemblyBegin(Vec vec)
  VecAssemblyEnd(Vec vec)
  MatAssemblyBegin(Mat mat, MatAssemblyType type)
  MatAssemblyEnd(Mat mat, MatAssemblyType type)
  type: either MAT_FLUSH_ASSEMBLY or MAT_FINAL_ASSEMBLY
  Vector and matrix are ready to use only after the assembly functions have been called.

• Vector operations (see next slides)
• Matrix operations (see next slides)

• PETSc documentation: http://www.mcs.anl.gov/petsc/documentation/index.html
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecAXPY(Vec y, PetscScalar a, Vec x);</td>
<td>$y = y + a \times x$</td>
</tr>
<tr>
<td>VecAYPX(Vec y, PetscScalar a, Vec x);</td>
<td>$y = x + a \times y$</td>
</tr>
<tr>
<td>VecWAXPY(Vec w, PetscScalar a, Vec x, Vec y);</td>
<td>$w = x + a \times y$</td>
</tr>
<tr>
<td>VecAXPBY(Vec y, PetscScalar a, PetscScalar b, Vec x);</td>
<td>$y = a \times x + b \times y$</td>
</tr>
<tr>
<td>VecScale(Vec x, PetscScalar a);</td>
<td>$x = a \times x$</td>
</tr>
<tr>
<td>VecDot(Vec x, Vec y, PetscScalar *r);</td>
<td>$r = \overline{x} \times y$</td>
</tr>
<tr>
<td>VecTDot(Vec x, Vec y, PetscScalar *r);</td>
<td>$r = \overline{x} \times y$</td>
</tr>
<tr>
<td>VecNorm(Vec x, NormType type, PetscReal *r);</td>
<td>$r =</td>
</tr>
<tr>
<td>VecSum(Vec x, PetscScalar *r);</td>
<td>$r = \sum x_i$</td>
</tr>
<tr>
<td>VecCopy(Vec x, Vec y);</td>
<td>$y = x$</td>
</tr>
<tr>
<td>VecSwap(Vec x, Vec y);</td>
<td>$y = x \text{ while } x = y$</td>
</tr>
<tr>
<td>VecPointwiseMult(Vec w, Vec x, Vec y);</td>
<td>$w_i = x_i \times y_i$</td>
</tr>
<tr>
<td>VecPointwiseDivide(Vec w, Vec x, Vec y);</td>
<td>$w_i = x_i/y_i$</td>
</tr>
<tr>
<td>VecMDot(Vec x, int n, Vec y[], PetscScalar *r);</td>
<td>$r[i] = \overline{x^j} \times y^j_i$</td>
</tr>
<tr>
<td>VecMTDot(Vec x, int n, Vec y[], PetscScalar *r);</td>
<td>$r[i] = \overline{x^j} \times y^j_i$</td>
</tr>
<tr>
<td>VecMAXPY(Vec y, int n, PetscScalar *a, Vec x[]);</td>
<td>$y = y + \sum_i a_i \times x[i]$</td>
</tr>
<tr>
<td>VecMax(Vec x, int *idx, PetscReal *r);</td>
<td>$r = \max x_i$</td>
</tr>
<tr>
<td>VecMin(Vec x, int *idx, PetscReal *r);</td>
<td>$r = \min x_i$</td>
</tr>
<tr>
<td>VecAbs(Vec x);</td>
<td>$x_i =</td>
</tr>
<tr>
<td>VecReciprocal(Vec x);</td>
<td>$x_i = 1/x_i$</td>
</tr>
<tr>
<td>VecShift(Vec x, PetscScalar s);</td>
<td>$x_i = s + x_i$</td>
</tr>
<tr>
<td>VecSet(Vec x, PetscScalar alpha);</td>
<td>$x_i = \alpha$</td>
</tr>
</tbody>
</table>
## PETSc matrix operations

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MatAXPY(Mat Y, PetscScalar a, Mat X, MatStructure);</td>
<td>$Y = Y + a \times X$</td>
</tr>
<tr>
<td>MatMult(Mat A, Vec x, Vec y);</td>
<td>$y = A \times x$</td>
</tr>
<tr>
<td>MatMultAdd(Mat A, Vec x, Vec y, Vec z);</td>
<td>$z = y + A \times x$</td>
</tr>
<tr>
<td>MatMultTranspose(Mat A, Vec x, Vec y);</td>
<td>$y = A^T \times x$</td>
</tr>
<tr>
<td>MatMultTransposeAdd(Mat A, Vec x, Vec y, Vec z);</td>
<td>$z = y + A^T \times x$</td>
</tr>
<tr>
<td>MatNorm(Mat A, NormType type, double *r);</td>
<td>$r =</td>
</tr>
<tr>
<td>MatDiagonalScale(Mat A, Vec l, Vec r);</td>
<td>$A = \text{diag}(l) \times A \times \text{diag}(r)$</td>
</tr>
<tr>
<td>MatScale(Mat A, PetscScalar a);</td>
<td>$A = a \times A$</td>
</tr>
<tr>
<td>MatConvert(Mat A, MatType type, Mat *B);</td>
<td>$B = A$</td>
</tr>
<tr>
<td>MatCopy(Mat A, Mat B, MatStructure);</td>
<td>$B = A$</td>
</tr>
<tr>
<td>MatGetDiagonal(Mat A, Vec x);</td>
<td>$x = \text{diag}(A)$</td>
</tr>
<tr>
<td>MatTranspose(Mat A, MatReuse, Mat * B);</td>
<td>$B = A^T$</td>
</tr>
<tr>
<td>MatZeroEntries(Mat A);</td>
<td>$A = 0$</td>
</tr>
<tr>
<td>MatShift(Mat Y, PetscScalar a);</td>
<td>$Y = Y + a \times I$</td>
</tr>
</tbody>
</table>
PETSc Krylov subspace solver

- **KSP**: Krylov subspace solver
- **PC**: preconditioner

Basic KSP functions:
- `KSPCreate(MPI_Comm comm, KSP *ksp)`
- `KSPSetOperators(KSP ksp, Mat Amat, Mat Pmat)` // assign the linear system to a KSP solver
- `KSPSetType(KSP ksp, KSPType type)` // KSP type: see next slides
- `KSPGetPC(KSP ksp, PC *pc)`
- `PCSetType(PC pc, PCTYPE type)` // PC type: see next slides
- `KSPSetTolerances(KSP ksp, PetscReal rtol, PetscReal abstol, PetscReal dtol, PetscInt maxits)`
- `KSPSolve(KSP ksp, Vec b, Vec x)`
- `KSPDestroy(KSP *ksp)`
### PETSc KSP types

<table>
<thead>
<tr>
<th>Method</th>
<th>KSPType</th>
<th>Options Database Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Richardson</td>
<td>KSPRICHARDSON</td>
<td>richardson</td>
</tr>
<tr>
<td>Chebyshev</td>
<td>KSPCHEBYSHEV</td>
<td>chebyshev</td>
</tr>
<tr>
<td>Conjugate Gradient [12]</td>
<td>KSPCG</td>
<td>cg</td>
</tr>
<tr>
<td>BiConjugate Gradient</td>
<td>KSPBICG</td>
<td>bicg</td>
</tr>
<tr>
<td>Generalized Minimal Residual [16]</td>
<td>KSPGMRES</td>
<td>gmres</td>
</tr>
<tr>
<td>Flexible Generalized Minimal Residual</td>
<td>KSPFGMRES</td>
<td>fgmres</td>
</tr>
<tr>
<td>Deflated Generalized Minimal Residual</td>
<td>KSPDGMRES</td>
<td>dgmres</td>
</tr>
<tr>
<td>Generalized Conjugate Residual</td>
<td>KSPGCR</td>
<td>gcr</td>
</tr>
<tr>
<td>BiCGSTAB [19]</td>
<td>KSPBCGS</td>
<td>begs</td>
</tr>
<tr>
<td>Conjugate Gradient Squared [18]</td>
<td>KSPCGS</td>
<td>cgs</td>
</tr>
<tr>
<td>Transpose-Free Quasi-Minimal Residual (1) [8]</td>
<td>KSPTFQMR</td>
<td>tfqmr</td>
</tr>
<tr>
<td>Transpose-Free Quasi-Minimal Residual (2)</td>
<td>KSPTCQMR</td>
<td>tcqmr</td>
</tr>
<tr>
<td>Conjugate Residual</td>
<td>KSPCR</td>
<td>cr</td>
</tr>
<tr>
<td>Least Squares Method</td>
<td>KSPLSQR</td>
<td>lsqr</td>
</tr>
<tr>
<td>Shell for no KSP method</td>
<td>KSPPREONLY</td>
<td>preonly</td>
</tr>
</tbody>
</table>
# PETSc PC types

<table>
<thead>
<tr>
<th>Method</th>
<th>PCType</th>
<th>Options Database Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jacobi</td>
<td>PCJACOBI</td>
<td>jacobj</td>
</tr>
<tr>
<td>Block Jacobi</td>
<td>PCBJACOBI</td>
<td>bjacobj</td>
</tr>
<tr>
<td>SOR (and SSOR)</td>
<td>PCSOR</td>
<td>sor</td>
</tr>
<tr>
<td>SOR with Eisenstat trick</td>
<td>PCEISENSTAT</td>
<td>eisenstat</td>
</tr>
<tr>
<td>Incomplete Cholesky</td>
<td>PCICC</td>
<td>icc</td>
</tr>
<tr>
<td>Incomplete LU</td>
<td>PCILU</td>
<td>ilu</td>
</tr>
<tr>
<td>Additive Schwarz</td>
<td>PCASM</td>
<td>asm</td>
</tr>
<tr>
<td>Algebraic Multigrid</td>
<td>PCGAMG</td>
<td>gamg</td>
</tr>
<tr>
<td>Linear solver</td>
<td>PCKSP</td>
<td>ksp</td>
</tr>
<tr>
<td>Combination of preconditioners</td>
<td>PCCOMPOSITE</td>
<td>composite</td>
</tr>
<tr>
<td>LU</td>
<td>PCLU</td>
<td>lu</td>
</tr>
<tr>
<td>Cholesky</td>
<td>PCCHOLESKY</td>
<td>cholesky</td>
</tr>
<tr>
<td>No preconditioning</td>
<td>PCNONE</td>
<td>none</td>
</tr>
<tr>
<td>Shell for user-defined PC</td>
<td>PCSHELL</td>
<td>shell</td>
</tr>
</tbody>
</table>
Exercise 4: Solve a linear system in parallel with PETSc

- **Task:** Compute the solution of a sparse-matrix linear system $Ax=b$, using a KSP solver (e.g. MINRES).

- **Solution:** C source code at /project/scv/examples/numlibs/petsc/ex42.c
  - Include petsc head file: `#include <petscksp.h>`
  - Call petsc functions: `KSPSetOperators`, `KSPSolve`, `KSPSetType`, etc.

- **Compile and run**
  - `module load petsc/3.7.0`  # set up PETSc
  - `make ex42`  # compile and link
  - `mpirun -n 24 ./ex42 -m 2400`  # run the job using 24 CPU cores
PETSc-dependent packages

• **SLEPc:**
  Scalable Library for Eigenvalue Problems

• **MOOSE:**
  Multiphysics Object-Oriented Simulation Environment finite element framework, built on top of libMesh and PETSc

More information:

6. GNU Scientific Lib: GSL

Main features:

• A numerical library for C and C++ programmers
• Provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting
• Uses an object-oriented design. Different algorithms can be plugged-in easily or changed at run-time without recompiling the program.
• It is intended for ordinary scientific users. Anyone who knows some C programming will be able to start using the library straight-away.
• Serial
Complete GSL subjects

- Mathematical Functions
- Complex Numbers
- Polynomials
- Special Functions
- Vectors and Matrices
- Permutations
- Combinations
- Multisets
- Sorting
- BLAS Support
- Linear Algebra
- Eigensystems

- Fast Fourier Transforms
- Numerical Integration
- Random Number Generation
- Quasi-Random Sequences
- Random Number Distributions
- Statistics
- Histograms
- N-tuples
- Monte Carlo Integration
- Simulated Annealing
- Ordinary Differential Equations
- Interpolation
- Numerical Differentiation

- Chebyshev Approximations
- Series Acceleration
- Wavelet Transforms
- Discrete Hankel Transforms
- One dimensional Root-Finding
- One dimensional Minimization
- Multidimensional Root-Finding
- Multidimensional Minimization
- Least-Squares Fitting
- Nonlinear Least-Squares Fitting
- Basis Splines
- Physical Constants
Exercise 5: Linear fit with GSL

Task: computes a least squares straight-line fit to a simple dataset, and outputs the best-fit line and its associated one standard-deviation error bars.
Solution for Exercise 5 in C

- C source code at /project/scv/examples/numlibs/gsl/linear_fit.c
  - Include gsl head file: #include <gsl/gsl_fit.h>
  - Call gsl function: gsl_fit_linear_est

**Compile and run**

```bash
module load gsl/1.16  # set up gsl environments
module show gsl/1.16  # show gsl environments

g++ -c linear_fit.c -I/share/pkg/gsl/1.16/install/include  # compile

g++ linear_fit.o -L/share/pkg/gsl/1.16/install/lib -static -lgsl -o linear_fit  # link to static libs

g++ linear_fit.o -L/share/pkg/gsl/1.16/install/lib -lgsl -lgslcblas -o linear_fit  # link to dynamic libs

./linear_fit  # run
```
More help?

BU Research Computing tutorial documents
http://www.bu.edu/tech/support/research/training-consulting/live-tutorials/

Submit jobs on BU SCC
http://www.bu.edu/tech/support/research/system-usage/running-jobs/submitting-jobs/

Send emails to us for questions
• help@scc.bu.edu
• shaohao@bu.edu