Numerical Libraries with C or Fortran

Shaohao Chen

Research Computing, IS&T, Boston University

Outline

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: $X_k \overset{\text{def}}{=} \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}, \quad k \in \mathbb{Z}$

❑ 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

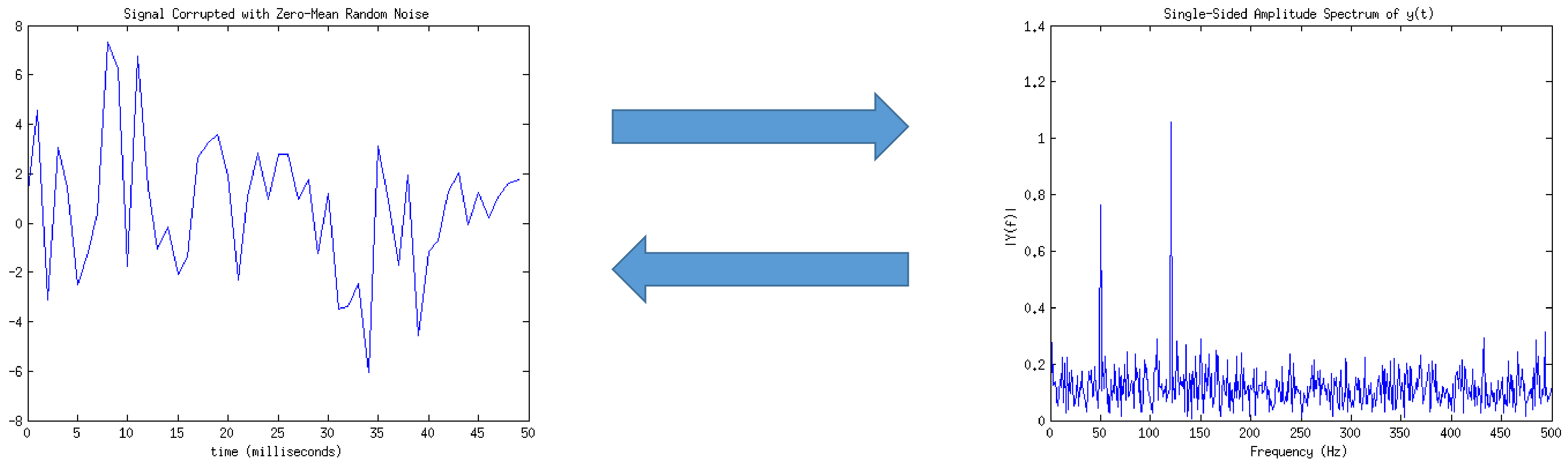
- fftw_plan_dft_r2c_1d, fftw_plan_dft_r2c_2d
- fftw_plan_dft_r2c_3d, fftw_plan_dft_r2c

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 moudle
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.

BLAS

- 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

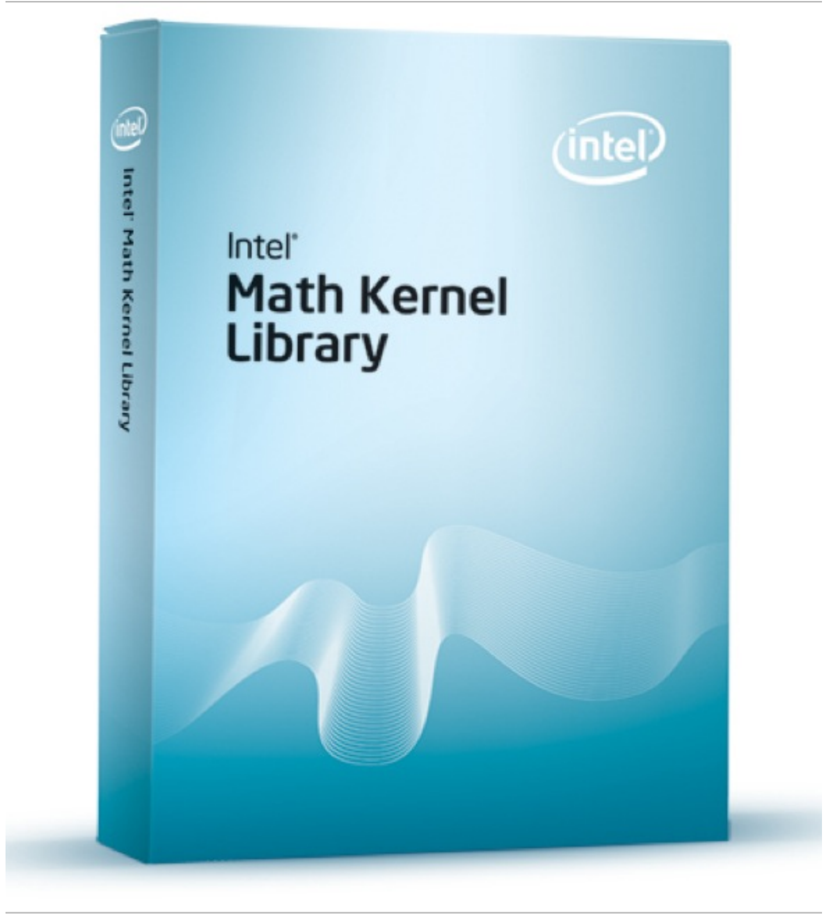
- ❑ Source code: /project/scv/examples/numlibs/lapack/matprod.f
- Initialize data for matrices A, B, C and real scalars alpha, beta.
- Call LAPACK function: DGEMM

❑ 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

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

? could be: s – single precision; d – double precision; c – single-precision complex; z – double-precision complex.

MKL LAPACK subroutines II

?heevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix using the Relatively Robust Representations.
?heevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a Hermitian matrix.
?hesv	Computes the solution to the system of linear equations with a Hermitian matrix A and multiple right-hand sides.
?posv	Computes the solution to the system of linear equations with a symmetric or Hermitian positive definite matrix A and multiple right-hand sides.
?syev	Computes all the eigenvalues and, optionally, the eigenvectors of a real symmetric matrix.
?syevd	Computes all the eigenvalues and, optionally, all the eigenvectors of a real symmetric matrix using a divide and conquer algorithm.
?syevr	Computes the selected eigenvalues and, optionally, the eigenvectors of a real symmetric matrix using the Relatively Robust Representations.
?syevx	Computes the selected eigenvalues and, optionally, the eigenvectors of a symmetric matrix.
?sysv	Computes the solution to the system of linear equations with a real or complex symmetric matrix $A$and multiple right-hand sides.

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

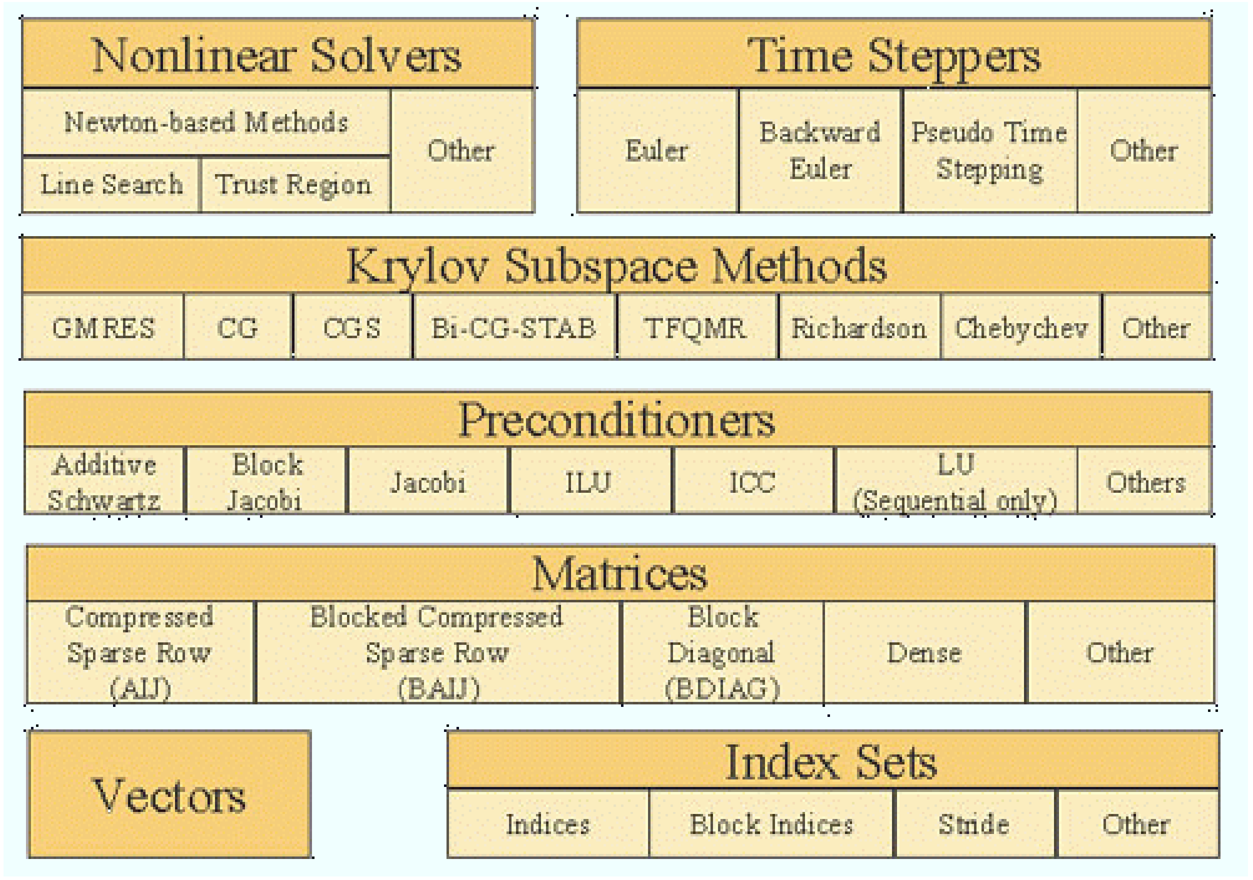
❑ Compile and run

```
module load intel/2016
icc -mkl dgesv_ex.c -o dgesv_ex
./dgesv
```

5. 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, 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



PETSc Basics I

- PetscInitialize   // call MPI_Initialize
- 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

  VecSetSizes(Vec v, PetscInt n, PetscInt N)   // local size n, global size N

  MatSetSizes(Mat A, PetscInt m, PetscInt n, PetscInt M, PetscInt N)   // local size m, n, global size M, N

- Set values of objects

  VecSetValues(Vec x, PetscInt ni, const PetscInt ix[], const PetscScalar y[], InsertMode mode)

  MatSetValues(Mat mat, PetscInt m, const PetscInt idxm[], PetscInt n, const PetscInt idxn[], const PetscScalar v[], InsertMode mode)   // Set values of a block. Unset blocks are filled with zero.

    mode: either INSERT_VALUES or ADD_VALUES

PETSc Basics III

- 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

PETSc vector operations

Function Name	Operation
VecAXPY(Vec y,PetscScalar a,Vec x);	$y = y + a * x$
VecAYPX(Vec y,PetscScalar a,Vec x);	$y = x + a * y$
VecWAXPY(Vec w,PetscScalar a,Vec x,Vec y);	$w = a * x + y$
VecAXPBY(Vec y,PetscScalar a,PetscScalar b,Vec x);	$y = a * x + b * y$
VecScale(Vec x, PetscScalar a);	$x = a * x$
VecDot(Vec x, Vec y, PetscScalar *r);	$r = \bar{x}' * y$
VecTDot(Vec x, Vec y, PetscScalar *r);	$r = x' * y$
VecNorm(Vec x,NormType type, PetscReal *r);	$r = \|\|x\|\|_{type}$
VecSum(Vec x, PetscScalar *r);	$r = \sum x_i$
VecCopy(Vec x, Vec y);	$y = x$
VecSwap(Vec x, Vec y);	$y = x$ while $x = y$
VecPointwiseMult(Vec w,Vec x,Vec y);	$w_i = x_i * y_i$
VecPointwiseDivide(Vec w,Vec x,Vec y);	$w_i = x_i / y_i$
VecMDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = \bar{x}' * y[i]$
VecMTDot(Vec x,int n,Vec y[],PetscScalar *r);	$r[i] = x' * y[i]$
VecMAXPY(Vec y,int n, PetscScalar *a, Vec x[]);	$y = y + \sum_i a_i * x[i]$
VecMax(Vec x, int *idx, PetscReal *r);	$r = \max x_i$
VecMin(Vec x, int *idx, PetscReal *r);	$r = \min x_i$
VecAbs(Vec x);	$x_i = \|x_i\|$
VecReciprocal(Vec x);	$x_i = 1/x_i$
VecShift(Vec x,PetscScalar s);	$x_i = s + x_i$
VecSet(Vec x,PetscScalar alpha);	$x_i = \alpha$

PETSc matrix operations

Function Name	**Operation**
MatAXPY(Mat Y, PetscScalar a,Mat X,MatStructure);	$Y = Y + a * X$
MatMult(Mat A,Vec x, Vec y);	$y = A * x$
MatMultAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A * x$
MatMultTranspose(Mat A,Vec x, Vec y);	$y = A^T * x$
MatMultTransposeAdd(Mat A,Vec x, Vec y,Vec z);	$z = y + A^T * x$
MatNorm(Mat A,NormType type, double *r);	$r = \|\|A\|\|_{type}$
MatDiagonalScale(Mat A,Vec l,Vec r);	$A = \text{diag}(l) * A * \text{diag}(r)$
MatScale(Mat A,PetscScalar a);	$A = a * A$
MatConvert(Mat A,MatType type,Mat *B);	$B = A$
MatCopy(Mat A,Mat B,MatStructure);	$B = A$
MatGetDiagonal(Mat A,Vec x);	$x = \text{diag}(A)$
MatTranspose(Mat A,MatReuse,Mat* B);	$B = A^T$
MatZeroEntries(Mat A);	$A = 0$
MatShift(Mat Y,PetscScalar a);	$Y = Y + a * I$

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

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

PETSc PC types

Method	**PCType**	**Options Database Name**
Jacobi	PCJACOBI	jacobi
Block Jacobi	PCBJACOBI	bjacobi
SOR (and SSOR)	PCSOR	sor
SOR with Eisenstat trick	PCEISENSTAT	eisenstat
Incomplete Cholesky	PCICC	icc
Incomplete LU	PCILU	ilu
Additive Schwarz	PCASM	asm
Algebraic Multigrid	PCGAMG	gamg
Linear solver	PCKSP	ksp
Combination of preconditioners	PCCOMPOSITE	composite
LU	PCLU	lu
Cholesky	PCCHOLESKY	cholesky
No preconditioning	PCNONE	none
Shell for user-defined PC	PCSHELL	shell

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:

http://www.mcs.anl.gov/petsc/index.html

http://www.mcs.anl.gov/petsc/publications/index.html

6. GNU Scientific Lib: GSL

Main features:

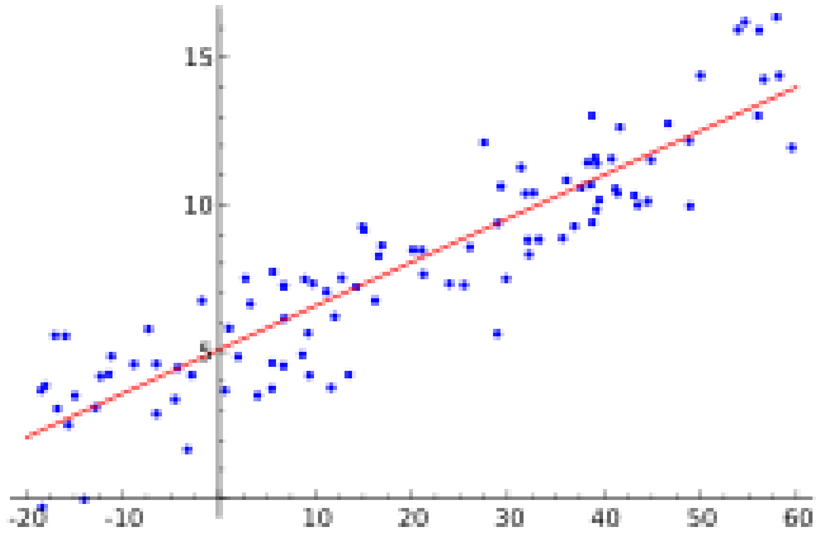
- A numerical library for C and C++ programmers
- Provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting
- Uses an object-oriented design. Different algorithms can be plugged-in easily or changed at run-time without recompiling the 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

```
module load gsl/1.16      # set up gsl environmens
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