Numerical Libraries

Ramses van Zon, Marcelo Ponce

2019 IHPCSS - Kobe, Japan

1

Don’t reinvent the wheel!

Modular programming

Modularity

- Scientific software can be large, complex and subtle.
- Interactions grow as (number of lines of code)$^{\mathbf{2}}$.
- You're either recoding the same thing, or are copy-pasting with large risk of mistakes.

Modular programming

Modularity

- Scientific software can be large, complex and subtle.
- Interactions grow as (number of lines of code)^{2}.
- You're either recoding the same thing, or are copy-pasting with large risk of mistakes.

How to

- Design the interface (header files in C++, interface in Fortran).
- Implementation in separate (ideally separate file).
- Enforce boundaries.

Modular programming

Modularity

- Scientific software can be large, complex and subtle.
- Interactions grow as (number of lines of code)^2.
- You're either recoding the same thing, or are copy-pasting with large risk of mistakes.

How to

- Design the interface (header files in C++, interface in Fortran).
- Implementation in separate (ideally separate file).
- Enforce boundaries.

Advantages

- It allows each module to be tested individually.
- It makes rebuilding software more efficient.
- It makes changing the code easier, and version control more powerful.

Modular programming, compiling and linking

- In modular programming, several object files for different modules need to be linked together.

- In the example below, thiscapp.cc/thisfapp.f90 contains the main program and alibrary.cc/alibrary.h/alibraryf.f90 are a c++ and f90 module.

Modular programming, compiling and linking

- In modular programming, several object files for different modules need to be linked together.

- In the example below, thiscapp.cc/thisfapp.f90 contains the main program and alibrary.cc/alibrary.h/alibraryf.f90 are a c++ and f90 module.

```
g++ -g -O2 -c -o thiscapp.o thiscapp.cc
g++ -g -O2 -c -o alibrary.o alibrary.cc
g++ -g thiscapp.o alibrary.o -o thiscapp
gfortran -g -O2 -c -o thisfapp.o thisapp.f90
gfortran -g -O2 -c -o alibraryf.o alibraryf.cc
gfortran -g thisfapp.o alibraryf.o -o thisfapp
```

(by the way, please use `make` or `cmake` in real life).

- What if we could use our alibrary in another project called newapp, without recompiling alibrary.c or alibraryf.f90?

From modular programming to libraries

From modular programming to libraries

- Copy .o and .h to separate directories:

```
alibrary.h      -> /base/include/alibrary.h
alibraryf.mod   -> /base/include/alibraryf.mod
alibrary.o      -> /base/lib/alibrary.o
alibraryf.o     -> /base/lib/alibraryf.o
```

From modular programming to libraries

- Copy .o and .h to separate directories:

```
alibrary.h      -> /base/include/alibrary.h
alibraryf.mod   -> /base/include/alibraryf.mod
alibrary.o      -> /base/lib/alibrary.o
alibraryf.o     -> /base/lib/alibraryf.o
```

- Must let compiler know where they are:
  Add `-I` flag for include directories.
  Absolute path for object file *(only for now!)*.

```
g++ -g -O2 -I/base/include -c -o thiscapp.o thiscapp.cc
g++ -g -o thiscapp thiscapp.o /base/lib/alibrary.o
gfortran -g -O2 -I/base/include -c -o thisfapp.o thisfapp.f90
gfortran -g -o thisfapp thisfapp.o /base/lib/alibraryf.o
```

Building with Libraries

Real libraries are similar; they have

- to be installed (and perhaps built first)
- header files (`.h` or `.hpp`) or module files (`.mod`) in some folder
- library files (object code) in a related folder.

Linux: library filenames start with `lib` and end in `.a` or `.so`.

```
g++ -g -o thiscapp thiscapp.o /base/lib/libalibrary.a
gfortran -g -o thisfapp thisfapp.o /base/lib/alibraryf.a
```

Instead of giving the explict path in linker command, we should specify:

- the path to the library's object using the `-L` option
- the object code using `-lNAME` (with a lower case letter l)
- libraries should come after the object files that use them.

```
g++ -g -L/base/lib -o thiscapp thiscapp.o -lalibrary
gfortran -g -L/base/lib -o thisfapp thisfapp.o -lalibraryf
```

More Notes on Libraries

- C++ standard libaries (`vector`,`cmath`,...) do not need any `-l...`'s.
- There are standard directories for libraries that needn't be specified in `-I` or `-L` options (`/usr/include`,...)
- Libraries installed through a package manager end up in standard paths; they just need `-l` options.
- You usually also do not need `-I` or `-L` for libraries accessed using the 'module load' command on the supercomputers.
- If you compile your own libraries in non-standard locations, you do need `-I` and `-L` options (as well as the `-lNAME` clause).

Example: MPI

```
$ mpicc -show
gcc -I/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/include
 -L/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib/release
 -L/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib
 -Xlinker --enable-new-dtags -Xlinker -rpath -Xlinker
 /opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib/release
 -Xlinker -rpath -Xlinker /opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib
 -lmpifort -lmpi -lrt -lpthread -Xlinker --enable-new-dtags -ldl
```

```
$ mpifc -show
gfortran -I/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/include/gfortran/4.8.0
 -I/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/include
 -L/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib/release
 -L/opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib
 -Xlinker --enable-new-dtags -Xlinker -rpath -Xlinker
 /opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib/release
 -Xlinker -rpath -Xlinker /opt/intel/compilers_and_libraries_2019.3.199/linux/mpi/intel64/lib
 -lmpifort -lmpi -lrt -lpthread -Xlinker --enable-new-dtags -ldl
```

Software modules

Software modules are a very common way to make software available on shared supercomputers.

```
$ module avail
------------------------------- /opt/modulefiles ------------------------------
abaqus/2016          abyss/2.0.2                       anaconda/4.2.0-3.5.2
abaqus/2017          AI/anaconda2-5.1.0_gpu            anaconda2/5.1.0
Abinit/7.10.5        AI/anaconda3-5.1.0_gpu            anaconda2/5.2.0
Abinit/8.0.8b        AI/anaconda3-5.1.0_gpu.2018-08    anaconda3/2019.03
Abinit/8.4.3         AIPS/31DEC16                      anaconda3/5.1.0
abyss/1.5.2          allpaths-lg/52488                 anaconda3/5.2.0(default)
...
```

E.g. try `module avail openblas` and `module help openblas`
(and `module spider openblas` for systems using lmod).

Must first do a `module load MODULE` before compiling and before running.

Installing libraries from source

What to do when your package manager does not have the library, or it's not in the software module stack, and you do not have permission to install packages in the system paths?

Installing libraries from source

What to do when your package manager does not have the library, or it's not in the software module stack, and you do not have permission to install packages in the system paths?

Compile from source code with a "base" or "prefix" directory.

Common installation procedure (but read documentation!):

```
$ ./configure --prefix=<BASE>
$ make
$ make install
```

```
$ cmake -DCMAKE_INSTALL_PREFIX=<BASE> ....
$ make
$ make install
```

You choose the `<BASE>`, but it should be a directory that you have write permission to, e.g., a subdirectory of your **$HOME**.

Using libraries that are not in standard directories

- For libraries that are not in standard directories, you need `-I<BASE>/include` and `-L<BASE>/lib` options in your compilation/link commands.

- Alternatively, you can omit these by setting some linux environment variables:

```
export CPATH="$CPATH:<BASE>/include"                    # compiler looks here for include files
export LIBRARY_PATH="$LIBRARY_PATH:<BASE>/lib"          # and here for library files
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:<BASE>/lib"    # runtime linker looks here
```

  You either enter these commands on the linux prompt before compiling, or, to set these automatically when you log in, add these lines to the `.bashrc` file in your home folder.

- The last one (`LD_LIBRARY_PATH`) may be necessary to run the application, even when it was successfully built and linked already.

- If the lbrary installs binary applications (i.e. commands) as well, you'll also need to set

```
export PATH="$PATH:<BASE>/bin"      # linux shell looks for executables here
```

2

So many libraries, so little time. . .

Some libraries for scientific computing (CPU)

BLAS interface for libraries for basic linear algebra operations. C C++ Fortran

LAPACK Linear solvers, eigenvalue problems, SVD, factorization. C C++ Fortran

ScaLAPACK Distributed version of LAPACK. C C++ Fortran

ARPACK-NG and SLEPc Eigenvalue problems. C C++ Fortran

BOOST Peer-reviewed portable C++ source libraries C++

Eigen Matrices, vectors, numerical solvers, . . . C++

FFTW Fourier and related transforms. C C++ Fortran

HDF5 and NetCDF Portable data model, library, and file formats. C C++ Fortran

GSL Numerical analysis library. C C++

PETSC Scalable (parallel) solution of partial differential equations. C C++ Fortran

Armadillo Matrix and vector maths similar to MATLAB. C++

Blaze Dense and sparse arithmetic. C++

Dlib Machine learning algorithms and tools. C++

Mlpack Machine learning algorithms. C++

Trilinos algorithms etc. for the solution of large-scale, complex multi-physics engineering and scientific problems. C C++

Some libraries for scientific computing (GPU/CUDA)

cuFFT Fast Fourier Transforms Library

cuBLAS Complete BLAS Library

cuSPARSE Sparse Matrix Library

cuRAND Random Number Generation Library

NPP Performance Primitives for Image & Video Processing

Thrust Templated C++ Parallel Algorithms & Data Structures

(from John Urbanic's accelerator slides).

Some libraries for scientific computing (Python)

numpy Faster arrays for Python. Mind all the lessons from the HPC Python programming session!

scipy Provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.

scikit learn (sklearn) Machine Learning in Python. Simple and efficient tools for data mining and data analysis.

pandas Tabular data manipulation and analysis.

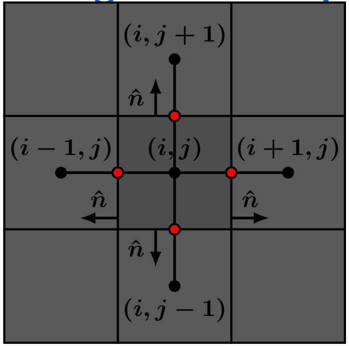
keras, tensorflow, theano, pytorch Neural nets/deep learning libraries.

Stan, PyMC3 Statistical modeling, data analysis, and Bayesian predictions.

3

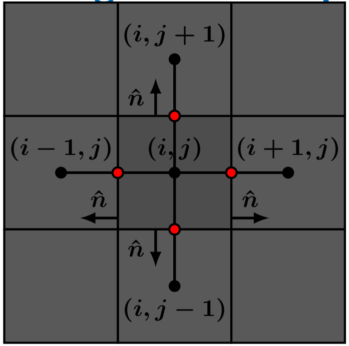
Example: Numerical Linear Algebra

I) Revisiting the heat-equation



$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

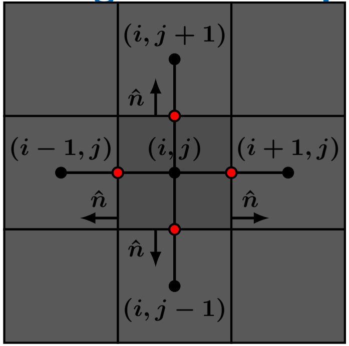
I) Revisiting the heat-equation



$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

1D: $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$

I) Revisiting the heat-equation

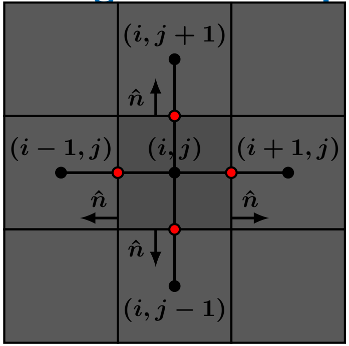


$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

1D: $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$
Discretizing the variable T in both time and space, with $T(t_i, x_j) = T_{i,j}$

$$\frac{\partial T_{i,j}}{\partial t} = \kappa \frac{\partial^2 T_{i,j}}{\partial x^2} \rightarrow \kappa \left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2} \right]$$

I) Revisiting the heat-equation



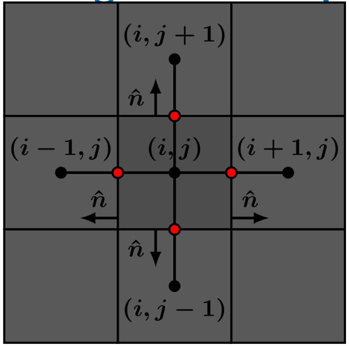
$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

1D: $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$
Discretizing the variable T in both time and space, with $T(t_i, x_j) = T_{i,j}$

$$\frac{\partial T_{i,j}}{\partial t} = \kappa \frac{\partial^2 T_{i,j}}{\partial x^2} \rightarrow \kappa \left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2} \right]$$

$$\begin{aligned} \Delta x^2 \frac{\partial T}{\partial t} &= \kappa \left[\left(\frac{T_{i,j} - T_{i-1,j}}{\Delta x} \right)(-1) + \left(\frac{T_{i+1,j} - T_{i,j}}{\Delta x} \right) + \right. \\ &\quad \left. \left(\frac{T_{i,j} - T_{i,j-1}}{\Delta x} \right)(-1) + \left(\frac{T_{i,j+1} - T_{i,j}}{\Delta x} \right) \right] \Delta x \\ &= \kappa \left(T_{i-1,j} + T_{i,j-1} + T_{i+1,j} + T_{i,j+1} - 4T_{i,j} \right) \end{aligned}$$

I) Revisiting the heat-equation



$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T$$

1D: $\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2}$
Discretizing the variable T in both time and space, with $T(t_i, x_j) = T_{i,j}$

$$\frac{\partial T_{i,j}}{\partial t} = \kappa \frac{\partial^2 T_{i,j}}{\partial x^2} \rightarrow \kappa \left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2} \right]$$

$$\begin{aligned}
\Delta x^2 \frac{\partial T}{\partial t} &= \kappa \left[\left(\frac{T_{i,j} - T_{i-1,j}}{\Delta x} \right)(-1) + \left(\frac{T_{i+1,j} - T_{i,j}}{\Delta x} \right) + \right. \\
&\quad \left. \left(\frac{T_{i,j} - T_{i,j-1}}{\Delta x} \right)(-1) + \left(\frac{T_{i,j+1} - T_{i,j}}{\Delta x} \right) \right] \Delta x \\
&= \kappa \left(T_{i-1,j} + T_{i,j-1} + T_{i+1,j} + T_{i,j+1} - 4T_{i,j} \right) \\
\Rightarrow T_{i,j} &= 0.25 \left(T_{i-1,j} + T_{i,j-1} + T_{i+1,j} + T_{i,j+1} \right)
\end{aligned}$$

$$\frac{\partial T_{i,j}}{\partial t} = k\left[\frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta x^2}\right]$$

If we write $T_{i,j}$ as a vector of values, we can rewrite our equation as a matrix operation. Note that there is one equation for each spatial point:

$$\frac{\partial}{\partial t}\begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix} = k \begin{bmatrix} & & & \vdots & & & \\ \cdots & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & 0 & \cdots \\ \cdots & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \cdots \\ \cdots & 0 & 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & \cdots \\ & & & \vdots & & & \end{bmatrix} \begin{bmatrix} \vdots \\ T_{i,17} \\ T_{i,18} \\ T_{i,19} \\ T_{i,20} \\ T_{i,21} \\ \vdots \end{bmatrix}$$

Which we can write as $\frac{\partial \vec{T}_i}{\partial t} = F\vec{T}_i$; $\alpha = \Delta t k/(\Delta x^2)$

```
#include <cmath>
#include <cblas.h>
int main() {
   double k = 0.2;
   double runtime = 2.0;
   int n = 100;
   double dx = 1.0 / (n - 1);
   double dt = 0.0005;
   int nsteps = runtime / dt;
   double a = 1.0, b = 0.0;
   double alpha = dt * k / dx**2;

   double *T = new double[n];
   double *rhs = new double[n];
   double **F = new double *[n];

   F[0] = new double[n * n];
   for(int i = 1; i < n; i++)
      F[i] = &F[0][i * n];
```

```
   for(int i = 0; i < n; i++) {
      T[i] = 0.0;    rhs[i] = 0.0;
      for(int j = 0; j < n; j++)
         F[i][j] = 0.0;}

   for (int i = 0; i < n; i++)
      if (i == 0) F[i][i] = 1.0;
      elseif ((i > 0) && (i < (n - 1))) {
         F[i][i-1] = alpha;  F[i][i+1] = alpha;
         F[i][i] = 1.0 - 2.0 * alpha;
      } elseF[i][i] = 1.0;

   for (int s = 1; i <= nsteps; s++) {
      double *temp = rhs;  rhs = T;  T = temp;
      rhs[0] = sin(dt * s * 10);
      rhs[n - 1] = 0.0;
      cblas_dgemv(CblasRowMajor, CblasNoTrans,
         n, n, a, F[0], n, rhs, 1, b, T, 1);
      // Output the result.
   }  // Deallocate.   return 0;}
```

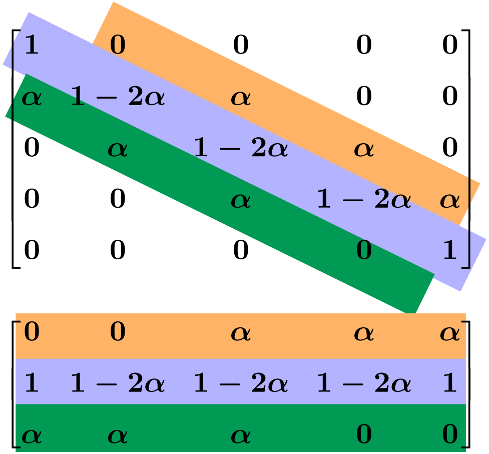
Notes about this implementation

A few things to recognize about this implementation:

- The code allocates and fills the entire matrix. Most of the matrix is zeros.
- This is a good way to implement things initially, for testing. It is not a good way to do things for production.
- Why? Because in this case the matrix is banded, and so a routine for a banded matrix should have been used to solve this problem.
- Who cares? Banded routines are faster and use much much less memory. Use them!
- Downside: banded matrices are a little more complicated to store, which is why we test them against the full matrix.
- The matrix in this example is also symmetric about the diagonal, so a banded-symmetric routine should have been used.

Banded-matrix storage

So how are banded matrices stored?



$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ \alpha & 1-2\alpha & \alpha & 0 & 0 \\ 0 & \alpha & 1-2\alpha & \alpha & 0 \\ 0 & 0 & \alpha & 1-2\alpha & \alpha \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & \alpha & \alpha & \alpha \\ 1 & 1-2\alpha & 1-2\alpha & 1-2\alpha & 1 \\ \alpha & \alpha & \alpha & 0 & 0 \end{bmatrix}$$

What about non-dense (i.e. sparse) matrices?

$$\begin{pmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 4 & & & & & \\ & 4 & -2 & 4 & & & & \\ & & 4 & -2 & 4 & & & \\ & & & 4 & -2 & 4 & & \\ & & & & & \ddots & & \\ & & & & & 4 & -2 & 1 \\ & & & & & & 1 & -2 \end{pmatrix}$$

Types

- Banded: `DGBSV`
- Tri-Diagonal: `DGTSV`
- Symmetric Positive Definite: `DPOSV`

II) Many computations reduce to linear algebra.

E.g. the Laplace equation from the challenge:

$$\Delta T = 0$$

with boundary conditions

$$T(0,y) = 0, T(x,0) = 0$$

$$T(x,1) = 100x, T(1,y) = 100y$$

II) Many computations reduce to linear algebra.

E.g. the Laplace equation from the challenge:

$$\Delta T = 0$$

with boundary conditions

$$T(0,y) = 0, T(x,0) = 0$$

$$T(x,1) = 100x, T(1,y) = 100y$$

Discretizing to an NxN grid gives

$$T_{i+1,j} + T_{i-1,j} + T_{i,j-1} + T_{i,j+1} - 4T_{ij} = 0$$

with boundary conditions

$$T_{0j} = 0, T_{i0} = 0$$

$$T_{i,N+1} = i/N, T_{N+1,j} = j/N$$

II) Many computations reduce to linear algebra.

E.g. the Laplace equation from the challenge:

$$\Delta T = 0$$

with boundary conditions

$$T(0,y) = 0, T(x,0) = 0$$

$$T(x,1) = 100x, T(1,y) = 100y$$

Discretizing to an NxN grid gives

$$T_{i+1,j} + T_{i-1,j} + T_{i,j-1} + T_{i,j+1} - 4T_{ij} = 0$$

with boundary conditions

$$T_{0j} = 0, T_{i0} = 0$$

$$T_{i,N+1} = i/N, T_{N+1,j} = j/N$$

We can rewrite this as a matrix equation

$$\sum_{i=0}^{N} \sum_{j=0}^{N} A_{klij} T_{ij} = 0$$

or, combining (i,j) to β and (k,l) to α,

$$\sum_{\beta} A_{\alpha\beta} T_{\beta} = 0$$

II) Many computations reduce to linear algebra.

E.g. the Laplace equation from the challenge:

$$\Delta T = 0$$

with boundary conditions

$$T(0,y) = 0, T(x,0) = 0$$

$$T(x,1) = 100x, T(1,y) = 100y$$

Discretizing to an NxN grid gives

$$T_{i+1,j} + T_{i-1,j} + T_{i,j-1} + T_{i,j+1} - 4T_{ij} = 0$$

with boundary conditions

$$T_{0j} = 0, T_{i0} = 0$$

$$T_{i,N+1} = i/N, T_{N+1,j} = j/N$$

We can rewrite this as a matrix equation

$$\sum_{i=0}^{N} \sum_{j=0}^{N} A_{klij} T_{ij} = 0$$

or, combining (i,j) to β and (k,l) to α,

$$\sum_{\beta} A_{\alpha\beta} T_{\beta} = 0$$

If we distinghuish interior and boundary β:

$$\sum_{interior\ \beta} A_{\alpha\beta} T_{\beta} = - \sum_{boundary\ \beta} A_{\alpha\beta} T_{\beta} \equiv b_{\alpha}$$

So we need to solve

$$AT = b$$

LAPACK Example: DGTSV (Solve $Ax = b$)

```
#include <iostream>
#include <lapacke.h>
const int N=5, NRHS=2;
int main(int argc, char **argv) {
    int ldb=N, info;
    double dl[N-1] = { 1, 4, 4, 1 };
    double d[N] = { -2, -2, -2, -2, -2 };
    double du[N-1] = {1, 4, 4, 1 };
    double b[N*NRHS] = {
        3, 5, 5, 5, 3,
        -1.56, 4.00, -8.67, 1.75, 2.86,
        9.81, -4.09, -4.57, -8.61, 8.99
    };
    info = LAPACKE_dgtsv(LAPACK_COL_MAJOR, N,
    NRHS, dl, d, du, b, ldb );
    ...
}
```

```
$ g++ -O2 dgesv.cc -o dgtsv
    -I${BLAS_INC} -L${BLAS_LIB} -lopenblas
$ ./dgtsv
```

```
Solutions ``x''
 -0.93   0.29  -6.10
  1.14  -0.99  -2.39
  2.05   0.43  -0.69
  1.14  -0.96   0.90
 -0.93  -1.91  -4.05
```

Example: GNU Scientific Library

Is a C library containing many useful scientific routines, such as:

- Root finding
- Minimization
- Sorting
- Integration, differentiation, interpolation, approximation
- Statistics, histograms, fitting
- Monte Carlo integration, simulated annealing
- ODEs
- Polynomials, permutations
- Special functions
- Vectors, matrices

Note: C library means we'll likely need to deal with some pointers and casts.

GSL root finding example

Suppose we want to find where $f(x) = a\cos(\sin(v + wx)) + bx - cx^2$ is zero.

```
// gslrx.cc
#include <iostream>
#include <gsl/gsl_roots.h>

struct Params {
 double v, w, a, b, c;
};
double examplefunction(double x, void* param){
 Params* p = (Params*)param;
 return p->a*cos(sin(p->v+p->w*x))+p->b*x-p->c*x*x;
}

int main() {
 double x_lo = -4.0;
 double x_hi = 5.0;
 Params args = {0.3, 2/3.0, 2.0, 1/1.3, 1/30.0};
 gsl_root_fsolver* solver;
 gsl_function      fwrapper;
 solver = gsl_root_fsolver_alloc(
             gsl_root_fsolver_brent);
```

```
 fwrapper.function = examplefunction;
 fwrapper.params = &args;
 gsl_root_fsolver_set(solver,&fwrapper,x_lo,x_hi);

 std::cout << "iter lower upper root err\n";

 int status = 1;
 for (int iter=0; status and iter < 100; ++iter) {
   gsl_root_fsolver_iterate(solver);
   double x_rt = gsl_root_fsolver_root(solver);
   double x_lo = gsl_root_fsolver_x_lower(solver);
   double x_hi = gsl_root_fsolver_x_upper(solver);
   std::cout << iter <<" "<< x_lo <<" "<< x_hi
             <<" "<< x_rt <<" "<<x_hi-x_lo<<"\n";
   status=gsl_root_test_interval(x_lo,x_hi,0,1e-3);
 }

 gsl_root_fsolver_free(solver);
 return status;
}
```

Compilation and linkage

- Lots of `gsl...` stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.

Compilation and linkage

- Lots of `gsl...` stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.
- How to compile on the command line?

Compilation and linkage

- Lots of `gsl...` stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.
- How to compile on the command line?

```
$ module load gcc gsl
$ export GSLINC=$GSL_HOME/include
$ export GSLLIB=$GSL_HOME/lib
$ g++ -O2  -c -I$GSLINC gslrx.cc -o gslrx.o
$ g++ gslrx.o -o gslrx -L$GSLLIB -lgsl -lgslcblas
```

Compilation and linkage

- Lots of `gsl...` stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.
- How to compile on the command line?

```
$ module load gcc gsl
$ export GSLINC=$GSL_HOME/include
$ export GSLLIB=$GSL_HOME/lib
$ g++ -O2  -c -I$GSLINC gslrx.cc -o gslrx.o
$ g++ gslrx.o -o gslrx -L$GSLLIB -lgsl -lgslcblas
```

On most clusters with software environment modules, you don't need the -I and -L options.

Compilation and linkage

- Lots of `gsl...` stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.
- How to compile on the command line?

```
$ module load gcc gsl
$ export GSLINC=$GSL_HOME/include
$ export GSLLIB=$GSL_HOME/lib
$ g++ -O2  -c -I$GSLINC gslrx.cc -o gslrx.o
$ g++ gslrx.o -o gslrx -L$GSLLIB -lgsl -lgslcblas
```

On most clusters with software environment modules, you don't need the -I and -L options.

Output

```
$ ./gslrx
iter lower      upper    root       err
 0    -4         -1.27657  -1.27657  2.72343
1    -1.95919  -1.27657  -1.95919  0.682622
2    -1.75011  -1.27657  -1.75011  0.473542
3    -1.75011  -1.74893  -1.74893  0.0011793
$
```

Compilation and linkage

- Lots of gsl... stuff.
- All of the algorithms come from the GSL.
- The rest is just wrappers, setting up parameters and calling the appropriate functions.
- There are pointers and typecasts, because we're dealing with a C library.
- How to compile on the command line?
- Even existing solutions like the once in the GSL, can't really be used until you understand the algorithm on a high level.

```
$ module load gcc gsl
$ export GSLINC=$GSL_HOME/include
$ export GSLLIB=$GSL_HOME/lib
$ g++ -O2  -c -I$GSLINC gslrx.cc -o gslrx.o
$ g++ gslrx.o -o gslrx -L$GSLLIB -lgsl -lgslcblas
```

On most clusters with software environment modules, you don't need the -I and -L options.

Output

```
$ ./gslrx
iter lower      upper     root       err
 0    -4         -1.27657  -1.27657  2.72343
1    -1.95919  -1.27657  -1.95919  0.682622
2    -1.75011  -1.27657  -1.75011  0.473542
3    -1.75011  -1.74893  -1.74893  0.0011793
$
```