C – Useful Resources

Paul Schrimpf

January 16, 2009x

Overview

- Numeric libraries
- Combining C and Fortran
- Optimization
- Automatic differentiation

General Resourse

- Linear Algebra – BLAS, LAPACK
- NetLib
- GAMS (Guide to Available Mathematical software)
- StatLib
- GSL (Gnu Scientific Library)
- List of Resources
- Commerical numerical libraries – NAG, IMSL
- Numerical Recipes – a convenient book, but not the best code
  - Critique of Numerical Recipes

Combining Fortan and C

- Many algorithms on Netlib are written in Fortran
- C and Fortran can be combined ... but it is not completely straightforward
- `f2c` translates Fortran to C – probably easiest way
- Other way: compile C to object file, compile Fortran to object file, link together
  - C and Fortran compilers follow slightly different conventions for names in object files – see Calling Fortran Subroutines from C/C++ for a workaround

Combining Fortran and C Example – Incomplete Gamma Function Derivatives

- Fortran code from StatLib

```
      SUBROUTINE DIGAMI(D, X, P, GPLOG, GP1LOG, PSIP, PSIP1, PSIDP,
     *      PSIDP1, IFAULT)
C
C     ALGORITHM AS 187  APPL. STATIST. (1982) VOL.31, NO.3
C     Computes derivatives of the incomplete gamma integral for positive
C     parameters, X, P, using a series expansion if P > X or X <= 1, and
C     a continued fraction expansion otherwise.
C     N.B. The user must input values of the incomplete gamma, digamma
C          and trigamma functions.  These can be obtained using AS 239
C          (or 32), AS 103 and AS 121 respectively.
C
      DOUBLE PRECISION X, P, GPLOG, GP1LOG, PSIP, PSIP1, PSIDP, PSIDP1
      DOUBLE PRECISION IFAULT

C     ... rest of code ommitted
```

Calling Code in C

```c
/* Function declaration                              *
 *  Note that FORTRAN_NAME is a macro to correct *
 *  name mangling; it is defined in fortran.h     */
void FORTRAN_NAME(digami)(double *d,const double *x,const double *p,
              const double *gplog, const double *gp1log,
              const double *psip, const double *psip1,
              const double *psidp,const double *psidp1,
              const double *ifault);
/*   ....   */

/* call fortran routine */
FORTRAN_NAME(digami)(d, &xb, &a, &gplog, &gp1log, &psip,
             &psip1, &psidp, &psidp1,&ifault);
```

- To compile:
  1. Compile C: `gcc -c callDigami.c -o callDigami.o`
  2. Compile Fortran: `g77 digami.f -c -o digami.o`
  3. Link: `gcc -lm digami.o callDigami.o -o program.exe`

Optimization

- Netlib opt/
- COIN-OR
- Global Optimization Software
    - ASA with Matlab Interface

Automatic Differentiation

- More developed in C than in Matlab
- ADIC
- ADOL-C
- CppAD
- FADBAD++ (recommended)

FADBAD++ – Function Preparation

- Function to differentiate: begin with `template <class T>`, replace `double` with `T`

```
template <class T> T safeLog(T x) {
  static double logDEL,iDEL,iDEL2;
  static double DEL = -1;
  if (DEL<0) {
    FILE *in = fopen(".DEL","r"); fscanf(in,"%lf",&DEL); fclose(in);
    printf("safeLog: DEL = %g\n",DEL);
    logDEL = log(DEL);     iDEL = 1./DEL;     iDEL2= iDEL*iDEL;
  }
  if (x>DEL) return(log(x));
  else return(logDEL - 1.5+2*(x)*iDEL - (x)*(x)*0.5*iDEL2);
}
```

FADBAD++ – Getting Derivatives

```
double score(double *grad, const double *i_x, const data *dta)
{
  int k;
  F<double> *x; // Initialize arguments
  x = new F<T>[dta->nParm]; // new is C++ simplification of malloc
  for(k=0;k<dta->nParm;k++) {
    x[k] = i_x[k];
    x[k].diff(k,dta->nParm); // tell FADBAD to record derivative wrt
                             // each x
  }
  F<double> f=likelihood(x,dta);  // Evaluate likelihood and derivatives
  for(k=0;k<dta->nParm;k++)
    grad[k] = f.d(k); // copy derivatives into grad
  delete[] x;
  return f.x();       // Return function value
}
```

Forward vs Reverse Automatic Differentiation

$$F : \Re^n \longrightarrow \Re^m$$

- Forward:
  - Applies chain rule to each step in calculation
  - Work grows with n, memory use constant
- Reverse:
  - Store calculation in memory, applies chain rule backwards
  - Work constant with n, memory use grows with n

Parallel Computing

- Idea: break a large problem into smaller pieces, distribute them among multiple processors / computers and gather result
- Plethora of languages and tools currently under development – Supercomputing and Parallel Computing Research Groups
- Single computer, multiple processors: OPENMP, libpthreads
- Multiple computers: PVM, MPI, Condor
- Matlab: Distributed Computing Toolbox, MatlabMPI, CONLAB

Parallel Computing – What's Different?

- Need to synchronize communication among processors / threads
  - Bad communication $\rightarrow$ no gain from multiple processors
  - Can cause difficult to diagnose bugs – e.g. The Therac 25 medical accelerator killed at least five people between 1985 and 1987 due to a race condition.
- Often, not too hard for economic applications
  - e.g. econometric problems with independent observations are "embarassinly parallel"

OpenMP

- Breaks program into multiple threads that can run concurrently on a single machine
- Pros:
  - Simple to use
  - Let's you fullow utilize all the processors and cores on a computer
- Cons:
  - Not supported by all compilers – works with icc, newer versions of gcc, and others (use -static option when want to compile on one machine and run on another)
  - Does not distribute the program to multiple computers
- Reference: https://computing.llnl.gov/tutorials/openMP/

OpenMP Basics

- OpenMP specific code in #ifdef OMP blocks, so that same code can be compiled with and without openmp

```
#ifdef OMP
# include <omp.h>
#endif
```

- OpenMP directives begin with #pragma omp
- Variable scope: private vs. shared

```
  double x = 0.0;
#pragma omp for default(shared) // compiler will automatically divide loop among t
  for(j = 0; j < n; j++) {
    double y = fn(input[j]);
#pragma omp critical
    { x += y; }
  } // could have also used the omp reduction commad
```

- Example: `ompex.c`

POSIX Threads

- Provide functionality similar to OpenMP
  - Less user-friendly
  - Available on nearly any computer (always for Linux and Unix, might need to install library for Windows)
- Also known as Pthreads
- Example: `pthreadex.c`

MPI

- **M**essage **P**assing **I**nterface – a simple library for communicating among processes
    - Each process has its own private memory and they only communicate through MPI commands
- Some commands:
    - MPI_Send() and MPI_Receive()
    - MPI_Bcast() – send a block of data from one thread to all other threads
    - MPI_Reduce() – take a block of data from all threads send it to one thread and combine it using some operation – e.g. add numbers from each thread together
    - Many more

MPI Example – main()

```c
int main(int argc, char *argv[])
{
  extern int rank, size;

#ifdef MPI
  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
#else
  rank=0;
  size=1;
#endif

  /* ... some more settup stuff omitted ... */

  /* call a minimizer */
  //sim_anneal(25.0,0.9,50,param,loglikelihood,0);
  dfpmin(param,1.0e-6,loglikelihood);

  if (rank==0) write_output(); // this way don't need to worry about mpi in io.c at al
#ifdef MPI
  MPI_Finalize();
#endif
}
```

MPI Example – loglikelihood()

```c
double loglikelihood(VECTOR param)
{
  double out,myout;
  int start, end;
  extern int rank, size;
#ifdef MPI
  MPI_Status status;
#endif
  /* divide the work */
  start = ((N_IND/size)*rank)+1;
  if (rank==size-1) end = N_IND;
  else  end = (N_IND/size)*(rank+1);

  myout=0.0;
  for(i=start;i<=end;i++) {
    aux=like_i(i);
    myout -= (aux<=TOO_SMALL ? LOG_TOO_SMALL : log(aux));
  }
#ifdef MPI
  MPI_Reduce(&myout,&out,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  MPI_Bcast(&out, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
#else
  out = myout;
#endif
  return(out);
}
```