AMath 483/583 — Lecture 11 — April 20, 2011

Today:

- Debugging Fortran
- Software packages
- zeroin for finding zeros of a function
- LAPACK and BLAS

Friday:

- Parallel computing concepts

Read: Class notes and references
There are several new sections!

Notes:

Optimizing Fortran

See the examples at

`$CLASSHG/codes/fortran/optimize.`

`$CLASSHG/codes/particles.`

Notes:

Developing programs to minimize bugs

- Start simple and add features slowly
  Tackle stripped-down version of problem first
- Modularize: break problem into pieces
  Subroutines or functions with
  well-defined inputs and outputs
  Develop and debug separately first

Notes:

Developing programs to minimize bugs

Unit tests: Test small pieces (early and often)

- Python has a unittest module to assist,
- Allows specification of test cases, test suites.

Regression testing:

Test that adding a new feature (or fixing a bug) didn't break old features.

Keep sample programs that test various features of the code, Run these after making improvements or "fixing" a bug.

Notes:

Debugging in Fortran

Need to compile with `-g` flag, no optimization.

(Runs slower, so recompile once debugged.)

gdb — command line debugger similar to pdb.

ddd — GUI front end for gdb, can be obtained on VM via:

```
$ sudo apt-get install ddd
```

Eclipse — IDE that uses gdb.

Much better commercial debuggers available, e.g. totalview.

Notes:

Debugging Fortran

See the examples at

`$CLASSHG/codes/fortran/debug.`

Notes:

Segmentation faults

Sometimes running a program gives:

```
$ ./a.out
Segmentation Fault
```

This generally means the code tried to write to a part of memory where it didn't have permission.

Or:

```
$ ./a.out
Bus error
```

This generally means a bad address not even in memory.

Often these are a result of an array index out of bounds.

Notes:

Segmentation faults

```
integer :: i
real(kind=8), dimension(10) :: x

do i=1,15
    x(i) = 20.d0
    print *, "i = ",i
    print *, x(i)
    enddo
```

produces:

```
 ...
 i =           10
   20.0000000000000
 i =   1077149696
Segmentation fault
```

Why? `x(11) points to memory where i is stored!`

Notes:

Overwriting variables

```
integer :: i
real(kind=8), dimension(10) :: x

do i=1,15
    x(i) = 0.d0
    print *, "i = ",i
    print *, x(i)
    enddo
```

Goes into an infinite loop — `i` gets reset to 0.

Notes:

Array bounds checking

```
$ gfortran -fbounds-check run1.f90
```

Gives:

```
...
 i =           10
   20.0000000000000
Fortran runtime error: Array reference out of bound
for array 'x', upper bound of dimension 1 exceeded
(in file 'demo1.f90', at line 11)
```


Notes:

Mathematical Software

It is best to use high-quality software as much as possible, for several reasons:

- It will take less time to figure out how to use the software than to write your own version. (Assuming it's well documented!)
- Good general software has been extensively tested on a wide variety of problems.
- Often general software is much more sophisticated that what you might write yourself, for example it may provide error estimates automatically, or it may be optimized to run fast.

Notes:

Software sources

- Netlib: **http://www.netlib.org**
- NIST Guide to Available Mathematical Software: **http://gams.nist.gov/**
- Trilinos: **http://trilinos.sandia.gov/**
- DOE ACTS: **http://acts.nersc.gov/**
- PETSc nonlinear solvers: **http://www.mcs.anl.gov/petsc/petsc-as/**
- Many others!

Notes:

Function `zeroin` from Netlib

The code in `$CLASSHG/codes/fortran/zeroin` illustrate how to use the function zeroin obtained from the Golden Oldies (go) directory of Netlib.

See: **http://www.netlib.org/go/index.html**

```
c     ============================================
      function zeroin(ax,bx,f,tol)
c     ============================================
      implicit double precision (a-h,o-z)
      external f
```

Note: Fortran 77 style!

Notes:

R.J. LeVeque, University of Washington AMath 483/583, Lecture 11, April 20, 2011

The BLAS

Basic Linear Algebra Subroutines

Core routines used by LAPACK (Linear Algebra Package) and elsewhere.

Generally optimized for particular machine architectures, cache hierarchy.

Can create optimized BLAS using ATLAS (Automatically Tuned Linear Algebra Software)

See notes and **http://www.netlib.org/blas/faq.html**

- Level 1: Scalar and vector operations
- Level 2: Matrix-vector operations
- Level 3: Matrix-matrix operations

Notes:

R.J. LeVeque, University of Washington AMath 483/583, Lecture 11, April 20, 2011

The BLAS

Subroutine names start with:

- S: single precision
- D: double precision
- C: single precision complex
- Z: double precision complex

Examples:

- DDOT: dot product of two vectors
- DGEMV: matrix-vector multiply, general matrices
- DGEMM: matrix-matrix multiply, general matrices
- DSYMM: matrix-matrix multiply, symmetric matrices

Notes:

R.J. LeVeque, University of Washington AMath 483/583, Lecture 11, April 20, 2011

LAPACK

Many routines for linear algebra.

Typical name: XYYZZZ

X is precision

YY is type of matrix, e.g. GE (general), BD (bidiagonal),

ZZZ is type of operation, e.g. SV (solve system),
EV (eigenvalues, vectors), SVD (singular values, vectors)

Notes:

Installing LAPACK

On Virtual Machine or other Debian or Ubuntu Linux:

```
$ sudo apt-get install liblapack-dev
```

This will include BLAS (but not optimized for your system).

Alternatively can download tar files and compile.

Notes:

Using libraries

If `program.f90` uses BLAS routines...

```
$ gfortran -c program.f90
$ gfortran -lblas program.o
```

or can combine as

```
$ gfortran -lblas program.f90
```

When linking together `.o` files, will look for a file called `libblas.a` (probably in `/usr/lib`).

This is a archived static library.

Notes:

Making blas library

Download **http://www.netlib.org/blas/blas.tgz**.

Put this in desired location, e.g. `$HOME/lapack/blas.tgz`

```
$ cd $HOME/lapack
$ tar -zxf blas.tgz    # creates BLAS subdirect
$ cd BLAS
$ gfortran -O3 -c *.f
$ ar cr libblas.a *.o  # creates libblas.a
```

To use this library:

```
$ gfortran -lblas -L$HOME/lapack/BLAS  \
           program.f90
```

Note: Non-optimized Fortran 77 versions.

Better approach would be to use ATLAS.

Notes:

Creating LAPACK library

Can be done from source at
http://www.netlib.org/lapack/

but somewhat more difficult.

Individual routines and dependencies can be obtained from e.g.:

http://www.netlib.org/lapack/double

Download `.tgz` file and untar into directory where you want to use them, or make a library of just these files.

Some routines are in
`$CLASSHG/codes/lapack/lapack-subset`.

Notes:

Memory management for arrays

Often a program needs to be written to handle arrays whose size is not known until the program is running.

Fortran 77 approaches:

- Allocate arrays large enough for any application,
- Use “work arrays” that are partitioned into pieces.

We will look at some examples from LAPACK since you will probably see this in other software!

The good news:

Fortran 90 allows dynamic memory allocation.

Notes:

DGESV — Solves a general linear system

http://www.netlib.org/lapack/double/dgesv.f

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV,
     $                  B, LDB, INFO )
```

`N` = size of system (square)

`A` = matrix on input, L,U factors on output,
`dimension(LDA,N)`

`LDA` = leading dimension of A
(number of columns in declaration of A)

```
real(kind=8) dimension(100,500) :: a
! fill a(1:20, 1:20) with 20x20 matrix
n = 20
lda = 100
```

Need this to index into `a(i,j) = (j-1)*lda + i`
(stored by columns)

Notes:

DGESV — Solves a general linear system

```
      SUBROUTINE DGESV( N, NRHS, A, LDA, IPIV,
     $                  B, LDB, INFO )
```

`NRHS` = number of right hand sides

`IPIV` = Returns pivot vector (permutation of rows)
`integer, dimension(N)`
Row `I` was interchanged with row `IPIV(I)`.

`B` = matrix whose columns are right hand side(s) on input
solution vector(s) on output.

`LDB` = leading dimension of `B`.

`INFO` = integer returning 0 if successful.

Notes:

Gaussian elimination as factorization

If A is nonsingular it can be factored as

$$PA = LU$$

where

P is a permutation matrix (rows of identity permuted),

L is lower triangular with 1's on diagonal,

U is upper triangular.

After returning from `dgesv`:
`A` contains L and U (without the diagonal of L),
`IPIV` gives ordering of rows in P.

Notes:

Gaussian elimination as factorization

Example:

$$A = \begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 6 \\ 2 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 6 \\ 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/2 & 1 & 0 \\ 1/2 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & 6 \\ 0 & 1.5 & 1 \\ 0 & 0 & 1/3 \end{bmatrix}$$

`IPIV` = (2,3,1)

and `A` ends up as

$$\begin{bmatrix} 4 & 3 & 6 \\ 1/2 & 1.5 & 1 \\ 1/2 & -1/3 & 1/3 \end{bmatrix}$$

Notes:

`dgesv` examples

See `$CLASSHG/codes/lapack/random`.

`randomsys1.f90` is with static array allocation.

`randomsys2.f90` is with dynamic array allocation.

Notes: