Numerical Libraries

Scientific Computing Sections 2.8, 3.8

Dr. Hyrum D. Carroll

October 4, 2016

Numerical Libraries

- People have devoted their lives to making efficient routines to solve

$$Ax = b$$

- The result of their work is a set of numerical libraries that can be used your program
- Often, there are versions in C, C++, Fortran, Java and other languages

Netlib

- One of the best sources for numerical libraries is `http://netlib.org`
- 600 million accesses to their website
- A good place to start and find example, codes, documentation, and libraries
- Most libraries have pre-compiled binaries that are available for common platforms

Using New Libraries

Pedagogical Philosophy

- If you give a man a fish, he eats for a day
- If you teach him how to fish, he has food for his life
- If you slap a man with a fish, he will be very, very confused. (Dr. John Wallin)

- I cannot teach you how to use 100 functions from each of 1000 libraries
- Instead, I will focus on how you can learn and use new library functions

Using New Libraries

- Try the examples from on-line sources
- Create a simple problem where you know the solution
- Prototype your solution in Matlab or Octave
  - *Get your algorithm working BEFORE you worry about libraries and syntax*
- Write the real code
- Debug it using the Matlab/Octave solution as your guide

- we would like a robust but standard routine - at least for now
- double precision
- appropriate for least squares

DGELS

Try Some Example Codes

Lapack Example from NAG

```
!      DGELS Example Program Text
!      NAG Copyright 2005.
!      .. Parameters ..

  integer, parameter :: kdble = selected_real_kind(15,307)

  integer, parameter ::  MMAX=16 ,NB=64 ,NMAX=8
  integer, parameter :: LDA=MMAX, LWORK=NMAX+NB*MMAX

!      .. Local Scalars ..
      real (kind=kdble) :: RNORM
      integer          I, INFO, J, M, N
!      .. Local Arrays ..
      real (kind=kdble) :: A(LDA,NMAX), B(MMAX), WORK(LWORK)
.
.
.
```

Sample Input Data

```
DGELS Example Program Data

  6      4                    :Values of M and N

 -0.57  -1.28  -0.39   0.25
 -1.93   1.08  -0.31  -2.14
  2.30   0.24   0.40  -0.35
 -1.93   0.64  -0.66   0.08
  0.15   0.30   0.15  -2.13
 -0.02   1.03  -1.43   0.50  :End of matrix A

 -2.67
 -0.55
  3.34
 -0.77
  0.48
  4.10                        :End of vector b
```

Sample Input Data

$$\begin{bmatrix} -0.57 & -1.28 & -0.39 & 0.25 \\ -1.93 & 1.08 & -0.31 & -2.14 \\ 2.30 & 0.24 & 0.40 & -0.35 \\ -1.93 & 0.64 & -0.66 & 0.08 \\ 0.15 & 0.30 & 0.15 & -2.13 \\ -0.02 & 1.03 & -1.43 & 0.50 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -2.67 \\ -0.55 \\ 3.34 \\ -0.77 \\ 0.48 \\ 4.10 \end{bmatrix}$$

Octave Solution

```
A =[  -0.57  -1.28  -0.39   0.25;
 -1.93   1.08  -0.31  -2.14 ;
  2.30   0.24   0.40  -0.35 ;
 -1.93   0.64  -0.66   0.08 ;
  0.15   0.30   0.15  -2.13 ;
 -0.02   1.03  -1.43   0.50 ]

b = [-2.67  -0.55   3.34  -0.77    0.48    4.10   ] '

x = A \ b

x =
   1.533874
   1.870748
  -1.524070
   0.039183
```

Sample Results

```
DGELS Example Program Results

Least squares solution
     1.5339      1.8707     -1.5241      0.0392

Square root of the residual sum of squares
     2.22E-02
```

Comments

- We do NOT need to use a square matrix
- We do NOT need to use the Normal equations method

Linking to Libraries

After the library is installed, you need to link to it

```
gfortran example.f90 -llapack
```

This will link to a library file name "liblapack.a" or "liblapack.so". (On the Mac, this is actually "liblapack.dyn".)

Sometimes you will need to specify the subdirectory where the library is found

```
gfortran example.f90 -L/usr/lib -llapack
```

The "-L" tells the compiler to look in the /usr/lib directory

Prototyping a Known Solution

Generating Data in Octave

```
n = 4;
m = 25;
a1 = 0.3e0;
a2 = -2.0e0;
a3 = 0.05e0;
a4 = -0.75e0;

for i = 1:m
  x(i) = i/10.0e0;
  y(i) = a1 + a2*x(i) + a3*x(i)**2 + a4*x(i)**3;
end
```

Solving the Problem in Octave

```
a = zeros(m,n);
for i = 1:m
   a(i, 1) = 1;
   a(i, 2) = x(i);
   a(i, 3) = x(i)**2;
   a(i, 4) = x(i)**3;
end


b = y;
sol = a\b';
sol(1:4)
```

The Solution

Does this make sense?

```
> sol(1:4)

ans =

   0.300000
  -2.000000
   0.050000
  -0.750000

>
```

Solutions

- makedata.f90
- linsq2.f90

Prototype Normal Equations Method

```
clear a, b;
a = zeros(n,n);
for col = 1: n
  for row = 1:n
    for i = 1:m
      a(col, row) = a(col, row) + x(i)**(col-1) * x(i)**(row-1);
    end
  end
end
```

Prototype Normal Equations Method

```
b = zeros(1,n);
for row = 1: n
  for i = 1:m
  b(row) = b(row) + y(i) * x(i)**(row-1);
  end
end

soln = a\b'
```

Prototype - Normal Equations Method

```
> soln = a\b'
soln =

    0.300000
   -2.000000
    0.050000
   -0.750000
```

- Octave

```
a = zeros(n,n);
for col = 1: n
  for row = 1:n
    for i = 1:m
      a(col, row) = a(col, row) + x(i)**(col-1) * x(i)**(row
    end
  end
end
```

- Fortran

```
a = 0.0d0
do col = 1, n
  do row = 1,n
    do i = 1,m
      a(col, row) = a(col, row) + x(i)**(col-1) * x(i)**(row
     enddo
  enddo
enddo
```

- Octave

```
soln = a\b'
```

- Fortran

```
call DGELS('No transpose', n, n, 1, A, LDA, &
           b, n, WORK, LWORK, INFO)
```