MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF MECHANICAL ENGINEERING
CAMBRIDGE, MASSACHUSETTS 02139

2.29 NUMERICAL FLUID MECHANICS— SPRING 2007

Problem Set 2

Posted 02/25/07, due Thursday 4 p.m. 03/8/07, Focused on Lecture 4 to 7

Problem 2.1 (6% of final grade): Advance your programming skills and review root finding methods

Review MATLAB help about:

- Function handle
- eval
- nargin
- varargin
- cell: as a data type
- switch: as flow control command
- fprintf
- lower

Here we want to develop a script as a generalized one dimensional solver. Later you can use it for next problems. The function that you write should provide the maximum ease of use, as well as the maximum amount of flexibility and adjustment. To that end and to develop a user friendly program:

- The function should have default values for everything so that the user can run it with minimum number of inputs.
- The input function (to be solved) should be either a function handle or a string (like '3*x^3-5*x+1').
- The program should have a nice command line output or plot displaying the gradual progress of solution.
- The user should be able to adjust/provide the below options, if necessary. Note that user should not need to memorize any order for them and option names should not be case sensitive:

  a) Method: Newton, Secant, Bi-Section, False-Position, Modified False-Position
  b) Initial guess: it can be two numbers for methods like Bi-Section
  c) Derivative of f (note that you can compute the derivative if user provides you with a string as solution equation)
  d) Absolute tolerance on x or f

e) Relative error on x
f) Maximum number of iterations
g) Plot

Here is what function should be like:

[x_solution, x_iterations, f_iterations] = solver(func_name)
[x_solution, x_iterations, f_iterations] = solver(func_name, x_guess)
[x_solution, x_iterations, f_iterations] = solver(func_name, x_guess, OPTIONS)

A few example calls are shown:

```
[x_f]=solver('x-sin(x)');
[x_f]=solver(@func);   % where func is a function handle

[x_f,x_it,f_itr]=solver(@func);
[x_f,x_it,f_itr]=solver(@func,2);

[x_f,x_it,f_itr]=solver(@func,[-2 2],'method','Bi-Section');
[x_f,x_it,f_itr]=solver(@func,[-2 2],'MEthod','bi-secTION');

[x_f,x_it,f_itr]=solver(@func,[-2 2],'plot','off');
[x_f,x_it,f_itr]=solver(@func,[-2 2],'method','Bi-Section','plot','off');
[x_f,x_it,f_itr]=solver(@func,[-2 2],'plot','off','method','Bi-Section');

[x_f,x_it,f_itr]=solver('x-sin(x)',2,'method','Newton');
[x_f,x_it,f_itr]=solver(@func,2,'method','Newton','f_derivative',@d_func);

[x_f,x_it,f_itr]=solver('x-sin(x)',2,'max_iteration',10, 'method','Newton');
[x_f,x_it,f_itr]=solver(@func     ,2,'max_iteration',10,'abs_tolerance',1e-8,'rel_tolerance',1e-6,'plot','on');
```

After writing the program you have to UPLOAD IT ON THE COURSE WEBSITE and PRINT IT AS WELL. That's all you have to do for this problem. This will replace the MATLAB workshop assignment about MATLAB programming.

Problem 2.2 (10 Points): Examine your root finding script

We are interested to find the roots of:

$$f(x) = e^{x} \sin x + e^{-x} \cos^{2} 2x$$

a) How many roots does this equation have?
b) Can you approximate analytically some roots of this equation and characterize their type? Discuss.
c) Find all the roots where $|x| < 4$. Use above program and examine all the methods for any root. For each root use "plotyy" command and plot two things at the same figure:
   - x versus number of iterations
   - Relative error of x (with respect to the most trusted solution) versus number of iterations (use logarithmic scale if needed)
d) Repeat part c and find the fist two roots where $x > 20$.
e) Repeat part c and find the fist two roots where $x < -20$.

Problem 2.3 (65 Points): Textbook problem

Solve the below problems from "Chapara and Canale" textbook:

- 5.4, 5.9, 5.15, 5.17 (Use your previous program if you can, then copy and paste the results)
- 6.1, 6.11, 6.15, 6.16, 6.23
- EXTRA CREDIT: 6.25 (5 Points)
- 10.6, 10.9, 10.12, 10.14

Problem 2.4 (15 Points): Computation cost in MATLAB

Review MATLAB help about:

- profile
- tic
- toc

Here we want to investigate the computational cost of basic operations $\{+,-,\times,\div\}$ and some functions $\{x,|x|,\sqrt{x},x^2,x^3,\sin x,\cos x,\tan x,e^x,e^{-x},\ln x\}$ in the MATLAB program. To that end we generate a huge vector/matrix of random numbers and repeat the operation on them as long as to get consistent result. Please specify your computer speed, memory and MATLAB version. Try to provide as much consistency as possible. Also clear and free your memory as much as possible. In each case examine both single and double data type and also report the cost time normalized by your CPU clock time (for example for a 2 GHz computer multiply time by $2\times10^9\ 1/\sec$).

a) Report assignment cost (or function y=x). Compare cost of a scalar assignment with cost of matrix assignment (normalized by the number of elements).

b) Compare cost of basic operations for scalar. Then compare $+,-$ for the matrixes and also element wise $\times,\div$. Discuss.

c) Repeat part b for specified functions.

Problem 2.5 (15 Points): Computational cost in other language (EXTRA CREDIT)

Repeat previous problem for a basic language (like C++ or Fortan), but only on the scalars.

Problem 2.6 (10 Points): Computational cost of determinant evaluation

a) We want to have a recursive formula for computational cost of determinant evaluation by expansion of minors. Assume that for a matrix with size n the number of

multiplication and summation\subtraction is $S_{\pm}(n), M_{\times}(n)$. Now compute $S_{\pm}(n+1), M_{\times}(n+1)$ by a recursive formula.

b) Now notice that $S_{\pm}(1)=0, M_{\times}(1)=0$ and try to develop a closed formula (or an order estimation) for $S_{\pm}(n), M_{\times}(n)$.

c) Study the textbook Box 9.1. Ignore the pivoting and compute or estimate the computational cost of determinant evaluation by Gauss elimination.

d) Compare "b" and "c" and discuss.

Problem 2.7 (10 Points): Correct and effective implementation of numerical algorithms

For any numerical code we are concerned with three issues:

1. Generalization (scope of algorithm) and exceptions
2. Numerical stability
3. Effective and fast implementation

Here we have a very simple MATLAB code for solving a linear system with Gauss elimination and we are interested to investigate these issues for below code

```
function [x]=gauss(A,b) % A: n*n matrix, b: n*1 vector where Ax=b

Ab=[A b];
n=length(A);
for i=1:n-1
    for j=i+1:n
        m=-Ab(i,i)/Ab(j,i);
        for k=1:n+1
            Ab(j,k)=m*Ab(j,k)+Ab(i,k);
        end
    end
end
x=zeros(n,1);
for i=n:-1:1
    sum=Ab(i,n+1);
    for j=i+1:n

        sum=sum-x(j)*Ab(i,j);
    end
    x(i)=sum/Ab(i,i);
end
```

a) Is that mathematically correct? Does it work for all matrixes or some matrixes exist which it fails to solve.

b) Is that numerically stable? Briefly discuss possible improvements.

c) Is that written effectively? Can we increase its speed? Is that operating with the minimum number of operations? More specifically about MATLAB, can the code be vectorized?

d) (EXTRA CREDIT: 10 Points) Improve the above code according to your answer in previous part (print the code).