CS4220 PS 1

Due: Weds, Jan 28, 2015

1.(Ascher and Greif, Section 3.6, Problem 1) Apply the bisection routine `bisect` to find the root of

$$f(x) = \sqrt{x} - 1.1,$$

starting from the interval $[0, 2]$ with `atol=1.e-8`.

(a) How many iterations are required? Does the count match your expectation based on our convergence analysis?

Solution:

With the starting interval $[0, 2]$ ($a = 0$, $b = 2$), the absolute error, defined as the absolute value of the difference between the root x^* and x_n (the iterate at step n), can be estimated by

$$|x^* - x_n| < \frac{(b-a)}{2} 2^{-n} < \texttt{atol},$$

$$n = \left\lceil \log_2 \left(\frac{b-a}{2\ \texttt{atol}} \right) \right\rceil . \tag{1}$$

With $\texttt{atol}=10^{-8}$, we predict we need $n = 27$. This number is exactly the same as the number of iterations used by `bisect` because the code implements the number of iterations as given by Eq.(1).

(b) What is the resulting absolute error? Could the absolute error be predicted by our convergence analysis?

The absolute error at $n = 27$ is 7.450581×10^{-9}, this error is bounded from below by the tolerance `atol`. We can predict the bound of the absolute error at a given iteration number, but not the exact value of the error.

2. (Ascher and Greif, Section 3.6, Problem 4) Consider the function $g(x) = x^2 + \frac{3}{16}$.

(a) This function has two fixed points. What are they?

$g(x)$ is continuous in $(-\infty, \infty)$. In order to find fixed points, we need to find intervals $[a, b]$ for which $a \leq g(x) \leq b$ for all $x \in [a, b]$.

It is straightforward to find "a" such that

$$g(a) = a^2 + \frac{3}{16} \geq a,$$

$$a^2 - a + \frac{3}{16} = \left(a - \frac{1}{4}\right)\left(a - \frac{3}{4}\right) \geq 0,$$

so

$$a \geq 3/4 \quad \text{or} \quad a \leq 1/4.$$

Similary, we find "b" such that

$$g(b) = b^2 + \frac{3}{16} \leq b,$$

so we have

$$1/4 \leq b \leq 3/4.$$

We can easily find the two intervals for the fixed points $[0, 1/4]$ and $[1/4, 3/4]$. The two fixed points are obtained when $g(a) = a$ and $g(b) = b$, hence they are $x_1^* = 1/4$ and $x_2^* = 3/4$, respectively.

(b) Consider the fixed point iteration $x_{k+1} = g(x_k)$. For which fixed point found in (a) can you be sure that the iteration will converge to that fixed point?

The derivative $g'(x) = 2x$ exists in these intervals. And we find the magnitude of the derivative near $x_1^* = 1/2$ to be

$$|g'(x)| \leq \frac{1}{2} = \rho < 1 \quad \text{for all } x \in [0, 1/4].$$

According to the Fixed Point Theorem, fixed point iteration using this g will converge to $x_1^* = 1/4$. However, near $x_2^* = 3/4$,

$$|g'(x)| \leq \frac{3}{2} = \rho > 1 \quad \text{for all } x \in [1/4, 3/4].$$

According to the Fixed Point Theorem, the fixed point iteration using this g will not converge to the root $x_2^* = 3/4$. The fixed point iteration code actually diverges if the initial guess $x_0 > 3/4$. (If $0 < x_0 < 3/4$, it will converge to $1/2$.)

(c) For $x_1^* = 1/4$, the convergence error at the k-th iteration can be estimated by

$$\frac{|x_k - x_1^*|}{|x_{k-1} - x_1^*|} \approx \rho^k \approx 0.1.$$

The number of iterations k to reduce the convergence error by a factor of 10, with $\rho = 1/2$, is

$$k = -\left[\frac{1}{\log_{10} \rho}\right] = 4.$$

3. (Ascher and Greif, Section 3.6, Problem 5) Write a MATLAB script computing the cubic root of a number $x = \sqrt[3]{a}$, with only basic arithmetic operations using Newton's method by finding the root of the function $f(x) = x^3 - a$. Run your program for $a = 0, 2, 10$. For each of the cases, start with a guess reasonably close to the solution. As a stopping criterion, require the function value whose root you are searching to be smaller than 10^{-8}. Print out the values of x_k and $f(x_k)$ in each iteration. Comment on the convergence rates and explain how they match your expectations.

Solution:

For $a = 0$, starting guess $x_0 = 1$. x in the last line is also the value of the root returned by the Newton's code.

```
i = 1   x=6.666667e-01    fx=1.000000e+00
i = 2   x=4.444444e-01    fx=2.962963e-01
i = 3   x=2.962963e-01    fx=8.779150e-02
i = 4   x=1.975309e-01    fx=2.601229e-02
i = 5   x=1.316872e-01    fx=7.707347e-03
i = 6   x=8.779150e-02    fx=2.283658e-03
i = 7   x=5.852766e-02    fx=6.766395e-04
i = 8   x=3.901844e-02    fx=2.004858e-04
i = 9   x=2.601229e-02    fx=5.940319e-05
i = 10  x=1.734153e-02    fx=1.760095e-05
i = 11  x=1.156102e-02    fx=5.215095e-06
i = 12  x=7.707347e-03    fx=1.545213e-06
i = 13  x=5.138231e-03    fx=4.578410e-07
i = 14  x=3.425487e-03    fx=1.356566e-07
i = 15  x=2.283658e-03    fx=4.019455e-08
i = 16  x=1.522439e-03    fx=1.190949e-08
i = 17  x=1.522439e-03    fx=3.528739e-09
====================
```

For $a = 2$, starting guess $x_0 = 1$,

```
i = 1   x=1.333333e+00    fx=-1.000000e+00
i = 2   x=1.263889e+00    fx=3.703704e-01
i = 3   x=1.259933e+00    fx=1.895523e-02
i = 4   x=1.259921e+00    fx=5.925932e-05
i = 5   x=1.259921e+00    fx=5.852585e-10
====================
```

For $a = 10$, starting guess $x_0 = 3$,

```
i = 1   x=2.370370e+00     fx=1.700000e+01
i = 2   x=2.173509e+00     fx=3.318295e+00
i = 3   x=2.154602e+00     fx=2.679586e-01
i = 4   x=2.154435e+00     fx=2.324175e-03
i = 5   x=2.154435e+00     fx=1.800132e-07
i = 6   x=2.154435e+00     fx=1.776357e-15
=====================
```

For $a = 2$ and $a = 10$, the convergence speed is quadratic, doubling the number of significant digits at each iteration, whereas for the case of $a = 0$, Newton's method only gets linear convergence, this is due to the case of multiple root at 0 (for $x^3 = 0$).

The results from this simple exercise exactly meet our expectations for the performance of the Newton's method.

Listing of the code:

```
function [ x ] = fixedPoint(x0, TOL)
% Fixed point iterations for a g(x) (in x=g(x))
% to find the root of x-g(x)=0.
% x0: initial guess
% TOL: function tolerance for |x-g(x)| < TOL
%
 g = @(x) x^2+ 3/16;
 maxit = 100;

 for k=1:maxit
     x=g(x0);
     disp( sprintf('x=%e, Fx=%e', x, x-g(x)));
     if ( abs(x-g(x))< TOL)
         return;
     end
     x0=x;
 end
end
```

4. The dispersion relation for shallow water waves is

$$\omega^2 = k\left(g + \frac{T}{\rho}k^2\right)\tanh(kh)$$

where

$$\begin{aligned} h &= \text{water depth}, \\ k &= \text{spatial wave number } (2\pi \text{ / wave length}), \\ \omega &= \text{frequency } (2\pi \text{ / period}), \\ T &= \text{surface tension}, \\ \rho &= \text{mass density}, \\ g &= \text{gravitational acceleration}. \end{aligned}$$

For water at 25°C, $T/\rho = 7.2 \times 10^{-5}\ m^3/s^2$, and the acceleration due to gravity is $g = 9.8$ m/s^2. Assuming these values, write a code using Newton's method to find k given ω and h, assuming $kh \ll 1$. Your routine should take the form

function k = ps1water(omega, h)

Solution:

Move all terms to one side of the shallow water waves dispersion equation and call this equation $F(k) = 0$ for any given h and ω. We also need $F'(k)$ which is required by the Newton's method:

$$F(k) = -\omega^2 + k\left(g + \frac{T}{\rho}k^2\right)\tanh(kh) = 0, \tag{2}$$

$$\frac{dF(k)}{dk} = (g + \frac{T}{\rho}\,k^2)\,h\,\mathrm{sech}^2(kh) + (g + 3\,\frac{T}{\rho}\,k^2)\tanh(kh). \tag{3}$$

From the assumption of $kh \ll 1$, we simply initialize $k = 0.1/h$. This initial guess turns out to be pretty robust.

The MATLAB function `ps1water(omega,h)` takes ω (in $1/s$) and h (in m) as arguments and returns the spatial wave number k (in m^{-1}) satisfying the shallow water wave dispersion relation by using the Newton's method.

The code is listed in the following page:

```
function k=ps1water(omega, h)
%
% function k=ps1water(omega, h)
% finding k, the wave number, of the
% shallow water waves dispersion equation
% using Newton's method.
% This function returns k (wave number, nondimensional)
% by the given omega (frequency [1/s]) and h (depth [m]).
% TOL: tolerance for convergence: |F(k)| < TOL
% maxit: max number of Newton's iterations

TOL=1E-8;
maxit=100;
T_over_rho=7.2E-5;
g=9.8;

% anonymous functions defined for F and F':
func = @(k) k*(g+T_over_rho*k^2)*tanh(k*h) - omega^2;

func_dot = @(k) (g+3*T_over_rho*k^2)*tanh(k*h) ...
  + k*(g + T_over_rho*k^2)*sech(k*h)^2* h;
%
  k = 0.1/h;  % initial guess based on kh << 1
  maxit = 100; % maximum number of Newton's iterations

  for i=1:maxit
    if ( abs(func(k)) < TOL)
% we have got the root!
      return;
    else
      k = k - func(k)/func_dot(k);
      disp(sprintf('i=%d   x=%e    fx=%e', i, k, func(k)));
    end
  end
% something is wrong if we get here:
  disp('Something is wrong: quitting Newton''s method');
  k=NaN;
```