Selected Solutions to Assignment #5

Page 102, exercise 3. For this problem, and several below, I wrote this Newton code:

mynewt.m

```
function z = mynewt(f,df,x0,tol)
% MYNEWT  Use Newton's method to solve  f(x) = 0,  given a function
% f,  its derivative  df,  an initial guess  x0,  and an error
% tolerance  tol.   Example:  To solve  sin(x) = 0  using initial
% guess  x0 = 3  to get 12 digits of accuracy:
%   >> f = @(x) sin(x);  df = @(x) cos(x)
%   >> mynewt(f,df,3,1e-12)

x = x0;
for n=1:20
  xnew = x - f(x) / df(x);
  if abs(xnew - x) < tol   % known to be good stopping criterion
    break
  end
  x = xnew;
end
if n<20, fprintf('  [did n = %d steps for error < %.2e]\n',n,tol)
else, warning('MYNEWT did 20 iterations; answer may be inaccurate'), end
z = xnew;
```

I "compared to bisection" using an already-online code, which is also printed on **Selected Solutions to Assignment #4**:

`http://www.dms.uaf.edu/~bueler/mybisect.m`

(a) So now I ran the Newton solver, checked, then ran bisection with a guessed bracket:

```
>> f = @(x) 1 - 2 * x * exp(-x/2);  df = @(x) (x - 2) * exp(-x/2);
>> mynewt(f,df,0,1e-6)
  [did n = 5 steps for error < 1.00e-06]
ans =  0.714805912362771
>> f(ans)
ans =  6.43929354282591e-15
>> mybisect(f,0.5,1,1e-6)
  [doing n = 19 steps for error < 1.00e-06]
ans =  0.714806556701660
>> f(ans)
ans = -5.79248273346877e-07
```

We see that `mybisect.m` does gets the accuracy we want, but not much more. Our Newton solver `mynewt.m` basically gets all the digits right. A final way to check is to compare to the built in root-finding program:

```
>> help fzero
  ...              % see the help file to run it
>> fzero(f,0)
ans =  0.714805912362778
```

This agrees with what we got from Newton to about 15 digits.

(b) Very similar:

```
>> f = @(x) 5 - (1/x);  df = @(x) 1 / (x*x);
>> mynewt(f,df,1/4,1e-6)
  [did n = 5 steps for error < 1.00e-06]
ans =  0.200000000000000
>> mybisect(f,1/7,1/3,1e-6)
  [doing n = 18 steps for error < 1.00e-06]
ans =  0.199999854678199
```

Newton is, of course, right-on. Bisection achieves only the accuracy we asked for. It is right-on with a different initial bracket:

```
>> mybisect(f,0.1,0.3,1e-6)
  [doing n = 18 steps for error < 1.00e-06]
ans =  0.200000000000000
```

Why?

Page 112, exercise 3. Here we are given enough information to estimate the "coefficient" in square brackets, in the Newton error formula:

$$\alpha - x_{n+1} = \left[-\frac{f''(\xi_n)}{2f'(x_n)} \right] (\alpha - x_n)^2.$$

In our case with $|f''(x)| \le 3$ and $|f'(x)| \ge 1$, we have

$$|\alpha - x_{n+1}| = \frac{|f''(\xi_n)|}{2|f'(x_n)|} |\alpha - x_n|^2 \le \frac{3}{2} |\alpha - x_n|^2.$$

Thus if $|\alpha - x_0| \le (1/2)$, as we are told, we have:

$$\begin{aligned}
|\alpha - x_1| &\le \frac{3}{2}|\alpha - x_0|^2 \le \frac{3}{2}\left(\frac{1}{2}\right)^2 = \frac{3}{8}, \\
|\alpha - x_2| &\le \frac{3}{2}|\alpha - x_1|^2 \le \frac{27}{128} = 0.2109375, \\
|\alpha - x_3| &\le \frac{3}{2}|\alpha - x_2|^2 \le 0.06674194.
\end{aligned}$$

Pages 112–113, exercise 5. **(a)** I wrote a special purpose program, which works for this application of Newton's method and evaluates the ratio involved in defining "order p" convergence.

exer5newt.m

```
function exer5newt(p)
% EXER5NEWT  special-purpose code!

alpha = log(2);        % yes, we know the exact answer
```

```
x = 1;                    % initial guess is somewhere near log(2)
f = @(x) 2 - exp(x);
df = @(x) - exp(x);
for n=1:4
  xnew = x - f(x) / df(x);
  Rn = abs(alpha - xnew) / (abs(alpha - x))^p
  x = xnew;
end
```

Note that it does only four iterations of Newton's method because, by that stage, we already have 13 digit accuracy. Running with the $p = 2$ case gives this:

```
>> exer5newt(2)
Rn =  0.452552160811118
Rn =  0.492973066030464
Rn =  0.499850846788258
Rn =  0.499738448014104
```

What is the "correct value" of $|R_n|$? By the Newton error formula, it is the limit of the coefficient in the error formula, as x_n and ξ_n both get close to the limit $\alpha = \ln 2$; note that $f'(x) = -e^x$ and $f''(x) = -e^x$ here:

$$|R_n| = \frac{|\alpha - x_{n+1}|}{(\alpha - x_n)^2} = \frac{|f''(\xi_n)|}{2|f'(x_n)|} \to -\frac{|f''(\alpha)|}{2|f'(\alpha)|} = \frac{e^{\ln 2}}{2e^{\ln 2}} = \frac{1}{2}.$$

Thus the result of our program suggests that the Newton method is converging as expected, quadratically and with a constant that we understand.

(b) For instance you might try $p = 1.5$ and $p = 2$:

```
>> exer5newt(2.5)
Rn =  0.816965249705635
Rn =  2.38813433375223
Rn =  16.7070571136546
Rn =  789.662651583110
>> exer5newt(1.5)
Rn =  0.250688090256722
Rn =  0.101762468047449
Rn =  0.0149548102538500
Rn =  3.16259754621637e-04
```

In neither case is R_n stabilizing. Rather, in the first case $R_n \to \infty$, apparently, and in the later case $R_n \to 0$. This is just what we expect because we have a proof (already) that Newton's method leads to a sequence $\{x_n\}$ that converges to α quadratically. Thus the ratio R_n should only converge to $0 < c < \infty$ with $p = 2$.

Page 113, exercise 6. Here we use the Newton error estimate with $f'(x) = 4 + \sin x$, $f''(x) = \cos x$, and thus

$$\begin{aligned}|\alpha - x_{n+1}| &= \frac{|f''(\xi_n)|}{2|f'(x_n)|} |\alpha - x_n|^2 \le \frac{\max_{x\in[-2,2]} |\cos(x)|}{2 \min_{x\in[-2,2]} (4 + \sin x)|} |\alpha - x_n|^2 \le \frac{1}{2(4-1)} |\alpha - x_n|^2 \\ &= \frac{1}{6} |\alpha - x_n|^2.\end{aligned}$$

All we know about x_0 is $x_0 \in [-2, 2]$. The root α is in the same interval; we know that because $f(-2) < 0$ and $f(2) > 0$, so $[-2, 2]$ is a bracket. Thus

$$|\alpha - x_0| \le 4.$$

Thus

$$\begin{aligned}|\alpha - x_1| &= \frac{1}{6} |\alpha - x_0|^2 \le \frac{1}{6} 4^2 = \frac{8}{3}, \\ |\alpha - x_2| &= \frac{1}{6} |\alpha - x_1|^2 \le \frac{1}{6} \left(\frac{8}{3}\right)^2 = \frac{32}{27}, \\ |\alpha - x_3| &= \frac{1}{6} |\alpha - x_2|^2 \le \frac{1}{6} \left(\frac{32}{27}\right)^2 = 0.23411, \\ |\alpha - x_4| &= \frac{1}{6} |\alpha - x_3|^2 \le 0.0091346, \\ |\alpha - x_5| &= \frac{1}{6} |\alpha - x_4|^2 \le 1.3907 \times 10^{-5}, \\ |\alpha - x_6| &= \frac{1}{6} |\alpha - x_5|^2 \le 3.2234 \times 10^{-11}.\end{aligned}$$

Clearly the Newton method is converging, and it will get 10^{-8} accuracy in at most 6 steps.

Note that for bisection we would get this accuracy from the initial interval $a = -2, b = 2$ in

$$n \ge \frac{\log(b - a) - \log(10^{-8})}{\log 2} = 28.575$$

steps. That is to say, in $n = 29$ steps. (Recall Theorem 3.1.)

Page 113, exercise 7. Here is what I see in actual computation, using $x_0 = -2, -1, 0, 1, 2$:

```
>> f = @(x) 4*x-cos(x);  df = @(x) 4+sin(x);
>> mynewt(f,df,-2,1e-8)
  [did n = 5 steps for error < 1.00e-08]
ans =  0.242674680640890
>> mynewt(f,df,-1,1e-8)
  [did n = 5 steps for error < 1.00e-08]
ans =  0.242674680640890
>> mynewt(f,df,0,1e-8)
  [did n = 4 steps for error < 1.00e-08]
ans =  0.242674680640890
>> mynewt(f,df,1,1e-8)
  [did n = 4 steps for error < 1.00e-08]
ans =  0.242674680640890
>> mynewt(f,df,-2,1e-8)
  [did n = 5 steps for error < 1.00e-08]
ans =  0.242674680640890
```

Note that the "$|x_{n+1} - x_n| \leq 10^{-8}$" criterion in `mynewt.m` is satisfied in fewer than 6 steps, but 6 steps do suffice to give us the desired accuracy. Convergence seems to occur for any $x_0 \in [-2, 2]$.

Page 116, exercise 1. We can use Theorem 3.3, of course. We want to see if M times the initial error, is less than one. So we estimate:

$$M = \frac{\max_{x\in[2,3]} |f''(x)|}{2 \min_{x\in[2,3]} |f'(x)|} \leq \frac{5}{2 \cdot 3} = \frac{5}{6}.$$

If we take $x_0 = 5/2 = 2.5$ and the root α is in the interval $[2, 3]$ then we know that the initial error is at most half the length of the interval:

$$|\alpha - x_0| \leq \frac{1}{2}.$$

Thus

$$M|\alpha - x_0| \leq \frac{5}{6} \cdot \frac{1}{2} = \frac{5}{12} < 1.$$

By Theorem 3.3, Newton's method will converge. And that theorem says

$$|\alpha - x_n| \leq M^{-1} \left(M|\alpha - x_0|\right)^{2^n} \leq \frac{6}{5} \left(\frac{5}{12}\right)^{2^n}.$$

Trial and error finds what n gives error at most 10^{-4}:

```
>> format short e
>> n=1;  power=2^n;  (6/5)*(5/12)^power
ans =  2.0833e-01
>> n=2;  power=2^n;  (6/5)*(5/12)^power
ans =  3.6169e-02
>> n=3;  power=2^n;  (6/5)*(5/12)^power
ans =  1.0902e-03
>> n=4;  power=2^n;  (6/5)*(5/12)^power
ans =  9.9038e-07
```

As so often, $n = 4$ iterations of Newton's method is enough.

Page 116, exercise 2. In brief, $M \leq 3/4$ and $|\alpha - x_0| \leq 1/2$ so $M|\alpha - x_0| \leq 3/8$, so Newton's method will converge. And we see that, by Theorem 3.3,

$$|\alpha - x_4| \leq M^{-1} \left(M|\alpha - x_0|\right)^{2^4} \leq \frac{4}{3} \left(\frac{3}{8}\right)^{16} = 2.04 \times 10^{-7} < 10^{-6}.$$

Thus four iterations suffices to get 10^{-6} accuracy. By Theorem 3.1, bisection would require

$$n \geq \frac{\log(1 - 0) - \log(10^{-6})}{\log 2} = 19.93$$

steps, that is to say, $n = 20$ steps, to achieve this accuracy.

Page 119, exercise 1. **(a)** Here $5 = 0.3125 \times 2^4$ so $b = 0.3125$. Thus we can use Newton's method on $f(x) = x^2 - b = x^2 - 0.3125$:

```
>> f = @(x) x^2 - 0.3125;   df = @(x) 2*x;
>> mynewt(f,df,0.5,1e-12)
   [did n = 5 steps for error < 1.00e-12]
ans =    0.559016994374947
>> ans * 2^2,  sqrt(5)
```

```
ans =     2.23606797749979
ans =     2.23606797749979
```

Page 123, exercise 1. All you need to know about secant method for this problem is that formula (3.30) on page 121 *is* the secant method:

$$x_{n+1} = x_n - f(x)\left[\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}\right].$$

I made it a code, but you can easily do it "by-hand" at the MATLAB/OCTAVE command line, too:

secant0.m

```
% SECANT0  Special code to do three steps of secant method for
%   x^3 - 2 = 0
% using inital guesses  x0 = 0  and  x1 = 1.

x0 = 0
x1 = 1
f = @(x) x^3 - 2;
for n=1:3
  xnew = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
  x0 = x1;
  x1 = xnew;
end
```

Running it gave this:

```
>> secant0
x0 = 0
x1 =  1
xnew =  2
xnew =  1.14285714285714
xnew =  1.20967741935484
>> 2^(1/3)
ans =  1.25992104989487
```

If you continue you will see that after 8 steps you have 15 digit agreement of x_n with $\sqrt[3]{2} = 2^{1/3}$.

P3. I decided to write a completely "fair" that only uses elementary operations $+, -, \times, \div$ to compute nth roots $x^{1/n}$, where n is an integer. First there is the nth-power function which does only multiplication to compute x^n:

nthpow.m

```
function z = nthpow(x,n)
% NTHPOW  For integer  n >= 1,  compute x^n  using multiplication.
% Example:
%   >> nthpow(5,3),  5^3

if floor(n) ~= n, error('only works with integer n'), end
if n < 1, error('n >= 1 is required'), end

z = x;
```

```
for j = 1:n-1
  z = z * x;
end
```

For the initial point z_0 to solve $f(z) = z^n - x = 0$ there must be many ways to do it, but $z_0 = 1$ works for the cases we consider, $x > 0$ and $n \geq 1$. So my solution to the problem looks like this:

nthroot.m

```
function z = nthroot(x,n)
% NTHROOT  Use Newton's method to solve compute  x^(1/n),  the
% nth root of x, for integer n:  n = 1,2,3,4,...
% Uses only -,*,/, and no built-in power function.  Calls helper
% code NTHPOW.  Examples show agreement with built-in power:
%   >> format long
%   >> nthroot(5,2),  sqrt(5)
%   >> nthroot(1000,7),  1000^(1/7)
%   >> nthroot(0.00001,13),  0.00001^(1/13)

if floor(n) ~= n, error('only works with integer n'), end
if n < 1, error('n >= 1 is required'), end

tol = 1e-14;
zold = 1;              % a very simple formula for initial guess!
for j = 1:200
  z = zold - (nthpow(zold,n) - x) / (n * nthpow(zold,n-1));
  if abs(z - zold) < tol,  break,  end
  zold = z;
end
fprintf('  [did n = %d steps for error < %.2e]\n',j,tol)
```

P4. I started this problem by getting basic understanding by a plot

```
>> a = 1.25e-6;                 % ft^2 / sec
>> T0 = 10;                     % degrees C
>> T = -30;                     % ditto
>> t = 30 * 24 * 60 * 60;       % 30 days in seconds
>> x = 0:.01:10;
>> u = T + (T0 - T) * erf(x / (2 * sqrt(a * t)));
>> plot(x,u), grid on, xlabel('x  (ft)')
```

This gave Figure 1. From this plot, or just from plugging in a few "reasonable" values for the relevant depth, we see that there is a root near $x = 3$, that is, $u(\alpha, 30 \text{ days}) = 0$ has a solution of about $\alpha = 3$ feet. This is enough to get going in Newton's method, as long as we can correctly input the function and its derivative.

Therefore define T, T_0, a, t to be their fixed values above, and define

$$f(x) = T + (T_0 - T)\,\mathrm{erf}\left(\frac{x}{2\sqrt{at}}\right).$$

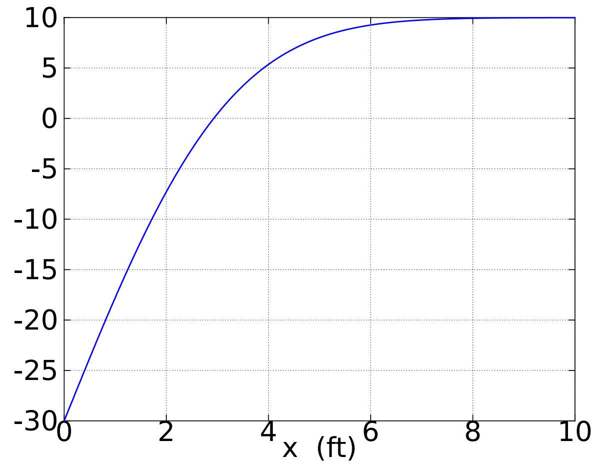

FIGURE 1. Plot of $u(x,t)$ for $t = 30$ days in problem P4.

From page 30,

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt$$

thus the derivative is

$$\frac{d}{dx}\left(\operatorname{erf}\right)(x) = \frac{2}{\sqrt{\pi}} e^{-x^2}.$$

(I have used the form of the Fundamental Theorem of Calculus which says that derivative and integral undo each other. Please *do* look it up it it is not natural!) So

$$f'(x) = (T_0 - T)\frac{2}{\sqrt{\pi}} e^{-(x/2\sqrt{at})^2} \frac{1}{2\sqrt{at}} = \frac{T_0 - T}{\sqrt{\pi a t}} \exp\left(-\frac{x^2}{4at}\right).$$

And now we can enter things into MATLAB/OCTAVE and use the `mynewt` which we already have:

```
>> at = 1.25e-6 * (30 * 24 * 60 * 60);
>> f = @(x) -30 + 40 * erf(x / (2 * sqrt(at)));
>> df = @(x) (40 / sqrt(pi * at)) * exp(- x.^2 / (4 * at));
>> mynewt(f,df,3.0,1e-12)
  [did n = 4 steps for error < 1.00e-12]
ans =  2.9283
>> f(ans)
ans = 0
```

We conclude that in this situation a pipe shallower than 2.9 feet or so might freeze.

Redoing this with $T = -50$ gives a danger-depth of $x = 3.5205$ feet. Redoing this with the original temperatures but a cold-snap of length 90 days gives $x = 5.0720$ feet. Thus colder temperatures, and longer cold periods, require deeper pipes. Not surprising.