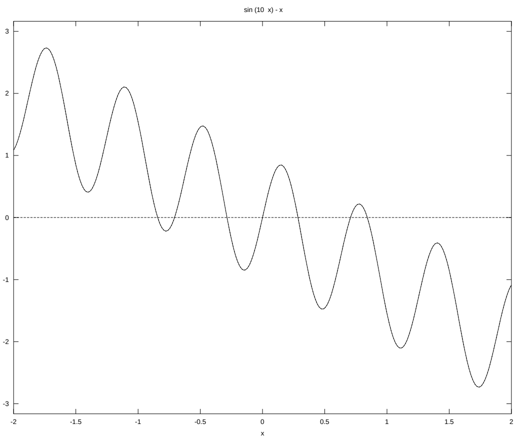
AMTH250 Assignment 7 Solutions

October 26, 2011

Question 1

A plot of the function shows 7 zeros.

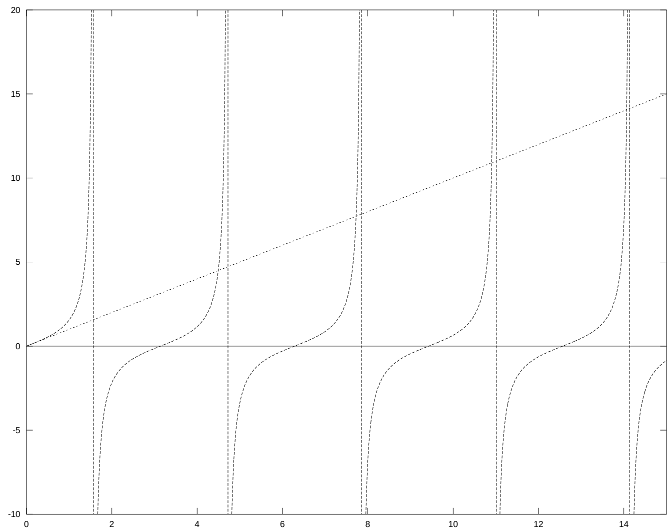


There is a zero at $x = 0$, the others are:

```
octave:> format long
octave:> f = @(x) sin(10*x) - x;
octave:> fzero(f,[-0.9 -0.8])
ans = -0.842320393236049
octave:> fzero(f,[-0.8 -0.6])
ans = -0.706817435809582
octave:> fzero(f,[-0.4 -0.2])
ans = -0.285234189445009
octave:> fzero(f,[0.2 0.4])
ans =  0.285234189445009
octave:> fzero(f,[0.6 0.8])
ans =  0.706817435809582
octave:> fzero(f,[0.8 0.9])
ans =  0.842320393236049
```

Question 2

Plotting $\tan x$ and x shows that the positive solutions of $x = \tan x$ lie in the intervals $[k\pi, (k+\frac{1}{2})\pi]$ for $k = 1, 2, 3, \ldots$ and get closer and closer to $(k+\frac{1}{2})\pi$ as k increases.



Therefore the millionth zero lies between $10^6\pi$ and $(10^6 + \frac{1}{2})\pi$. Because the solution is close to $(10^6 + \frac{1}{2})\pi$ and π is not exactly computer representable we need to check the bracketing of the solution.

```
octave:> f = @(x) tan(x)-x;
octave:> a = 1e6*pi;
octave:> b = 1e6*pi +pi/2;
octave:> f(a), f(b)
ans = -3141592.65358979
ans =  3518503757.55121
```

We have bracketed the solution

```
octave:> x0=fzero(f, [a b])
x0 =  3141594.22438580
```

Question 3

(a) The derivatives are

$$
\begin{aligned}
{g_1}'(x) &= \frac{2x}{3} & \qquad {g_1}'(2) &= \frac{4}{3} \\
{g_2}'(x) &= \frac{3}{2\sqrt{3x-2}} & \qquad {g_2}'(2) &= \frac{3}{4} \\
{g_3}'(x) &= \frac{2}{x^2} & \qquad {g_3}'(2) &= \frac{1}{2} \\
{g_4}'(x) &= \frac{2x^2-6x+4}{(2x-3)^2} & \qquad {g_3}'(2) &= 0
\end{aligned}
$$

For the first fixed point we expect divergence. For the second and third we expect linear convergence with rates of convergence 3/4 and 1/2 respectively. For the fourth we expect quadratic convergence.

(b) We use the function `iterate` to perform `n` iterations of $x_{k+1} = g(x_k)$ starting at `x0`:

```
function x  = iterate (g, x0, n)
  x = zeros(1,n+1)
  x(1) = x0
  for i = 1:n
    x(i+1) = g(x(i))
  end
endfunction
```

(1) First equation:

```
octave:> g1 = @(x) (x.^2 + 2)/3;
octave:> x1 = iterate(g1, 2.01, 15)
x1 =
 Columns 1 through 8:
   2.0100   2.0134   2.0179   2.0239   2.0321   2.0432   2.0582   2.0787
 Columns 9 through 16:
   2.1070   2.1465   2.2025   2.2837   2.4050   2.5947   2.9109   3.4911

octave:> x1 = iterate(g1, 1.99, 15)
x1 =
 Columns 1 through 8:
   1.9900   1.9867   1.9823   1.9765   1.9689   1.9589   1.9457   1.9286
 Columns 9 through 16:
   1.9065   1.8782   1.8426   1.7984   1.7447   1.6813   1.6090   1.5296
```

These are clearly diverging from the fixed point $x = 2$.

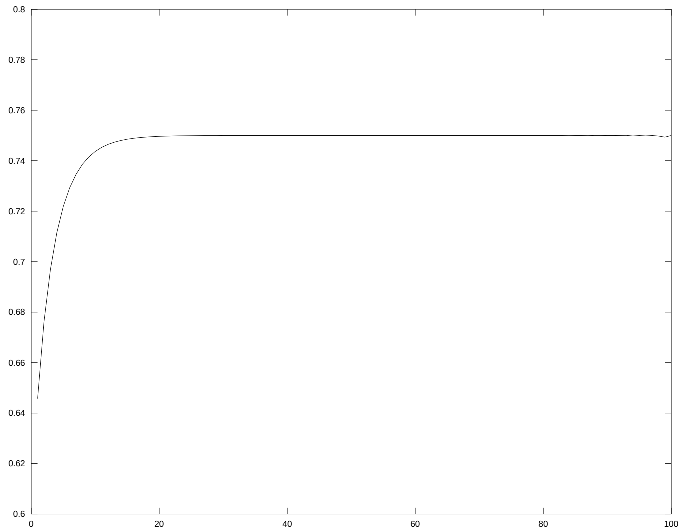
(2) Second equation:

```
octave:> g2 = @(x) sqrt(3*x-2);
octave:> x2 = iterate(g2, 3, 100)
x2 =
 Columns 1 through 8:
   3.0000   2.6458   2.4366   2.3043   2.2165   2.1563   2.1140   2.0837
 Columns 9 through 16:
   2.0618   2.0459   2.0341   2.0254   2.0190   2.0142   2.0106   2.0079
 Columns 17 through 24:
   2.0059   2.0045   2.0033   2.0025   2.0019   2.0014   2.0011   2.0008
 Columns 25 through 32:
   2.0006   2.0004   2.0003   2.0002   2.0002   2.0001   2.0001   2.0001
 Columns 33 through 40:
   2.0001   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000
```

This is clearly converging to $x = 2$.

We compute and plot the ratio of errors.

```
octave:> err2 = x2-2;
octave:> ratio2 = err(2:101)./err(1:100);
octave:> plot(ratio2)
```



The ratio of successive errors is clearly approaching $g'(2) = 3/4$ in accord with theory.

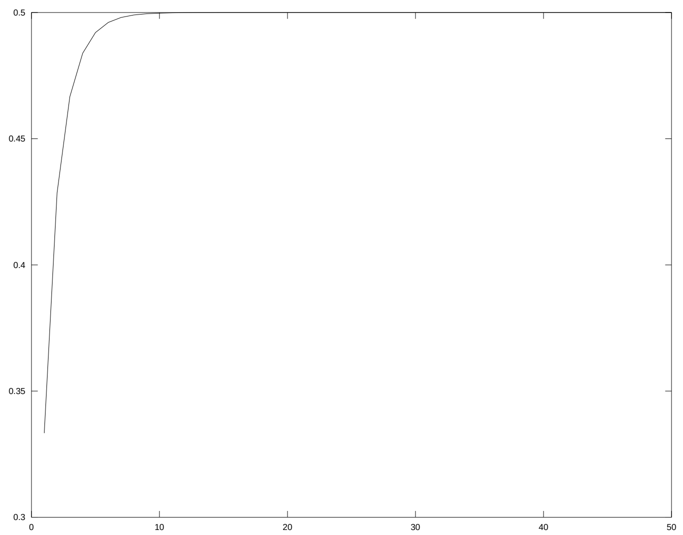
(3) Third equation:

```
octave:15> g3 = @(x) 3 - 2./x;
octave:16> x3 = iterate(g3, 3, 50)
x3 =
 Columns 1 through 8:
   3.0000   2.3333   2.1429   2.0667   2.0323   2.0159   2.0079   2.0039
 Columns 9 through 16:
   2.0020   2.0010   2.0005   2.0002   2.0001   2.0001   2.0000   2.0000
 Columns 17 through 24:
   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000
 Columns 25 through 32:
   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000
 Columns 33 through 40:
   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000   2.0000
```

Again this is clearly converging to $x = 2$.

We compute and plot the ratio of errors.

```
octave:> err3 = x3-2;
octave:> ratio3 = err3(2:51)./err3(1:50);
octave:> plot(ratio3)
```



The ratio of successive errors is clearly approaching $g'(3) = 1/2$ in accord with theory.

(4) Fourth equation:

```
octave:> g4 = @(x) (x.^2-2)./(2*x-3);
octave:> x4 = iterate(g4, 3, 10)
x4 =
 Columns 1 through 8:
   3.0000   2.3333   2.0667   2.0039   2.0000   2.0000   2.0000   2.0000
 Columns 9 through 11:
   2.0000   2.0000   2.0000
octave:30> err4 = x4-2
err4 =
 Columns 1 through 8:
   1.00000   0.33333   0.06667   0.00392   0.00002   0.00000   0.00000   0.00000
 Columns 9 through 11:
   0.00000   0.00000   0.00000
```

Here we have very rapid convergence. The 6th iterate is exactly 2.

Quadratic convergence can be tested by looking at the ratio of the error at one step to the *square* of the error at the previous step.

```
octave:> ratio4 = err4(2:6)./(err4(1:5).^2)
ratio4 =
   0.33333   0.60000   0.88235   0.99222   0.99997
```

This result is consistent with quadratic convergence, but we only have a limited number of iterations before the error becomes zero.

Question 4

(a) For

$$f(x) = \frac{1}{x} - y$$

Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

gives

$$x_{n+1} = x_n - \frac{1/x_n - y}{-1/{x_n}^2} = x_n + (x_n - y{x_n}^2) = 2x_n - y{x_n}^2$$

which is suitable for our purposes.

(b) With $y = 2$ our method is

$$x_{n+1} = 2x_n - 2{x_n}^2$$

We can use `iterate` from the previous question to examine the convergence of the method. With $x_0 = 0.9$ or $x_0 = 0.1$ we get convergence to 1/2:

```
octave:> nrecip = @(x) 2*x - 2*x.^2;
octave:> iterate(nrecip,0.9,10)
ans =
 Columns 1 through 8:
   0.90000   0.18000   0.29520   0.41611   0.48593   0.49960   0.50000   0.50000
 Columns 9 through 11:
   0.50000   0.50000   0.50000
octave:> iterate(nrecip,0.1,10)
ans =
 Columns 1 through 8:
   0.10000   0.18000   0.29520   0.41611   0.48593   0.49960   0.50000   0.50000
 Columns 9 through 11:
   0.50000   0.50000   0.50000
```

(i) With $x_0 = 1.1$ we have divergence:

```
octave:> iterate(nrecip,1.1,10)
ans =
 Columns 1 through 6:
   1.1000e+00  -2.2000e-01  -5.3680e-01  -1.6499e+00  -8.7442e+00  -1.7041e+02
 Columns 7 through 11:
  -5.8421e+04  -6.8261e+09  -9.3190e+19  -1.7369e+40  -6.0335e+80
```

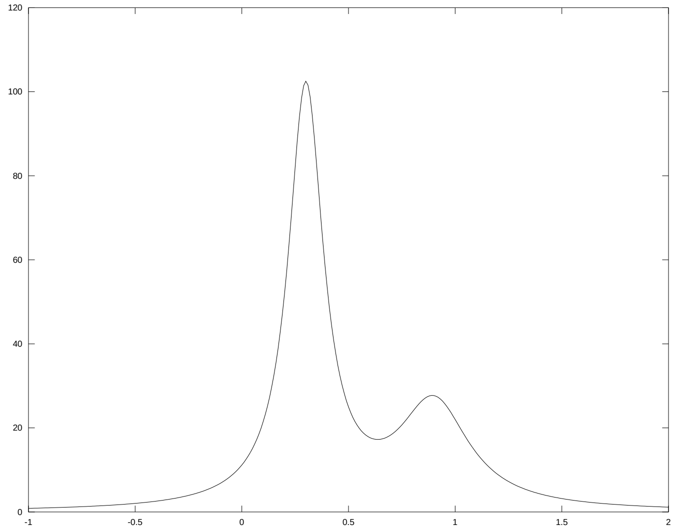
(ii) If we start at $x_0 = 1$ we get convergence to 0 in one step.

```
octave:> iterate(nrecip,1,10)
ans =
   1   0   0   0   0   0   0   0   0   0   0
```

(iii) Experimentation shows that the method converges to 1/2 for all initial guesses x_0 with $0 < x_0 < 1$.

Question 5

A plot of the function shows that the maximum occurs between $x = 0$ and $x = 0.5$ and that the function is unimodal between these values.



(a) Since we want the maximum of $f(x)$ we need to minimize $-f(x)$.

```
f = @(x) -1./((x-0.3).^2 +0.01) - 1./((x-0.9).^2+0.04);
```

For golden section we take a tolerance of 10^{-9}:

```
octave:> [xg1 xg2] = goldsec(f,0,0.5,1e-9)
xg1 =  0.300375620090181
xg2 =  0.300375620924802
```

For successive parabolic interpolation we take 10 iterations:

```
octave:24> xp = parab(f, 0, 0.25, 0.5, 10)
warning: division by zero
xp =
 Columns 1 through 4:
   0.263483965014577   0.343023032554397   0.300368952410001   0.300880960308783
 Columns 5 through 8:
   0.300343650724686   0.300375623559020   0.300375621962563   0.300375621869049
 Columns 9 and 10:
   0.300375621915806                 NaN
```

The division by zero occurs because two function values used by the method are equal. This indicates that the method has reached the limit of its accuracy.

Using `fminbnd`

```
octave:> xf = fminbnd(f,0,0.5)
xf =  0.300375621982956
```

(b) Note that value from `fminbnd` and the final value from `parab` do not lie in the interval returned from `goldsec`. To make sense of this we look at the values of $f(x)$ at these points:

```
octave:31> -f(xg1)
ans =  102.501408560372
octave:32> -f(xg2)
ans =  102.501408560372
octave:33> -f(xp(9))
ans =  102.501408560372
octave:34> -f(xf)
ans =  102.501408560372
```

Numerically there is a range of values at which $f(x)$ attains its maximum.

To find this range we compute $f(x)$ over a range of values and look for those at which $f(x)$ attains it maximum.

```
octave:> x1 = 0.300375620;
octave:> x2 = 0.300375640;
octave:> xx = linspace(x1, x2, 10001);
octave:> fxx = f(xx);
octave:> xmax = xx(fxx == min(fxx));
octave:> x1 = min(xmax)
x1 =  0.300375620698000
octave:> x2 = max(xmax)
x2 =  0.300375622420000
```

These last two values are the limits of the range over which $f(x)$ attains its maximum.

Taking the mean of these as our estimate of the maximum and half the difference as the error we get

```
octave:> x0 = mean([x1 x2])
x0 =  0.300375621559000
octave:> xerr = (x2-x1)/2
xerr =  8.60999993523848e-10
```

We find that the maximum is attained at

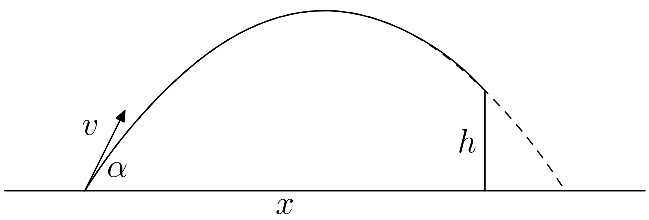
$$x_0 = 0.30037562156 \pm 1 \times 10^9$$

An accurate value, obtained by solving $f'(x) = 0$, is

$$x_0 = 0.300375621619754$$

Question 6

Here is the geometry of the problem. The path of the water is a parabola.



The equation relating x and α

$$\frac{g}{2v^2\cos^2(\alpha)}x^2 - \tan(\alpha)x + h = 0$$

is a quadratic in x and has a solution for x provided

$$\tan^2\alpha - \frac{2gh}{v^2\cos^2\alpha} \geq 0$$

or, on solving for α,

$$\sin^{-1}\left(\sqrt{\frac{2gh}{v^2}}\right) \leq \alpha < \frac{\pi}{2}$$

Note that for given α we take x to be the larger of the two roots of the quadratic.

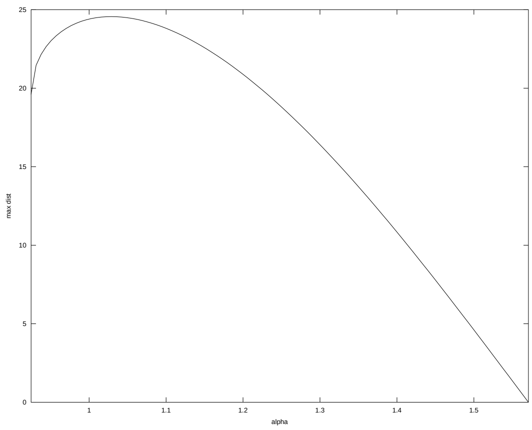
In Octave we need to construct x as function of α We do this in two steps, first constructing the quadratic for x as a function of α and then defining x as the larger root of the polynomial:

```
g = 9.80665;
v = 20;
h = 13;
amin = asin(sqrt(2*g*h/v^2)) + eps;  % The eps avoid complex
amax = pi/2 - eps;                   % roots due to rounding

p = @(a) [g./(2.*(v*cos(a)).^2) -tan(a) h];
x = @(a) max(roots(p(a)));
```

To plot x as a function we need a `for` loop since The function `x` cannot take a vector as its argument.

```
aa = linspace(amin, amax, 100);
xx = zeros(1,100);
for i = 1:100
  xx(i) = x(aa(i));
end
plot(aa,xx)
```



Now we can find the maximum distance x:

```
a0 = fminbnd(@(a) -x(a), amin, amax);
maxx = x(a0);

octave:> maxx
maxx =  24.5603132414681
```

So the maximum distance is

$$x = 24.56 \text{ m}$$