Homework 4

Ramaseshan Parthasarathy, Saurabh Prasad

Problem 1

1. The method does converge to f(x) = 0:
   Assume the method converges, knowing $\lim_{k\to\infty} x_k = a$, and $f'(a) \neq 0$:

   $a = \frac{a+a-\frac{f(a)}{f'(a)}}{2} \Rightarrow 2a = 2a - \frac{f(a)}{f'(a)} \Rightarrow \frac{f(a)}{f'(a)} = 0$

   $\therefore$ The method converges to f(x) = 0
2. This method does converge under the same conditions as Newton's method:

   $g(x) = x - \dfrac{f(x)}{2f'(x)} \Rightarrow g'(x) = 1 - \dfrac{f'^2(x) - f(x)f''(x)}{2f'^2(x)}$

   Three key assumptions were made in class notes with regards to Newton's Method:
   (1) $f(a) = 0$ (i.e. it is a solution to $f(x) = 0$)
   (2) $f'(a) \neq 0$, and
   (3) $f''(a)$ is bounded near a (i.e. f'' is continuous)
   Using the above assumptions, we arrive at the following: $\lim_{x\to a} g'(x) = \frac{1}{2}$
   $\therefore$ $g(x)$ must converge given it's a contraction on $(a-\delta, a+\delta)$
3. The order of convergence is 1:
   From the class notes, $g(x_k) = g(a) + g'(a)(x_k - a) + \frac{g''(c)}{2}(x_k - a)^2$
   This can be rewritten as $|e_{k+1}| = \frac{1}{2}|e_k| + \frac{g''(c)}{2}|e_k|^2, |e_{k+1}| \leq \mathrm{L}\,|e_k|$
   Moreover, since $|e_{k+1}| \leq \mathrm{L}\,|e_k|^d$, where $d = 2$, we have a first order convergence!

Problem 2

1. Analysis of convergence properties:

   $|g_1'(x)| = \dfrac{|2x|}{3} = \dfrac{4}{3} > 1$ (divergence)

   $|g_2'(x)| = \left|\dfrac{3}{2\sqrt{3x-2}}\right| = \dfrac{3}{4} < 1$ (linear convergence with constant 0.75)

   $|g_3'(x)| = \left|\dfrac{2}{x^2}\right| = \dfrac{1}{2} < 1$ (linear convergence with constant 0.50)

   $|g_4'(x)| = \left|\dfrac{-2(x^2-2)}{(2x-3)^2} + \dfrac{2x}{2x+3}\right| = 0$ (quadratic convergence)

   Showing linear convergence is equivelent to saying g_i is a contraction.

2. Verifying my analysis:

```
def fixedp(f,x0,eps=10e-6,n=100):
    """ Fixed point algorithm """
    e = 1
    itr = 0
    xp = []
    x = 2.5
    while(e > eps):
        x = f(x0)
```

```
        #print(x)
        e = np.linalg.norm(np.fabs(x-x0))
        x0 = x
        xp.append(x0)
        #print(xp[itr])
        itr = itr + 1
        print(itr)
    return x,xp

def main():
    g1 = lambda x: (pow(x, 2) + 2)/3
    g2 = lambda x: math.sqrt(3*x - 2)
    g3 = lambda x: 3 - 2/x
    g4 = lambda x: (pow(x,2)-2)/(2*x - 3)

    x,xp = fpiters(g1, 2.1, 100)

    print(x)

if __name__ == "__main__":
    main()
```

The results came out as expected.

Problem 3

The code for Bisection, Secant, and Newton methods for solving 1-D nonlinear equations is given below:

```
import numpy as np
import math
maxIterations=1000000
threshold=0.00001
def f1(x):
        return x**3-2*x-5.
def f1prime(x):
        return 3.*x**2-2
def f2(x):
        return math.exp(-x)-x
def f2prime(x):
        return -math.exp(-x)-1
def f3(x):
        return x*math.sin(x)-1.
def f3prime(x):
        return x*math.cos(x)+math.sin(x)
def f4(x):
        return x**3-3.*x**2+3.*x-1.
def f4prime(x):
        return 3*x**2-6*x+3

def Newton(f,fprime,x0):
        x=x0
        sol=np.zeros(100) #Create array of computed solutions
        sol[0]=x
        xnew=x-f(x)/fprime(x) # Compute first iteration
        iterations=1
        sol[iterations]=xnew
        while(math.fabs(x-xnew)>threshold and iterations<maxIterations):
                x=xnew
                xnew=x-f(x)/fprime(x)
                iterations+=1
                sol[iterations]=xnew

        ek=np.zeros(iterations);
        for i in range(iterations):
```

```
            ek[i]=math.log10(math.fabs(xnew-sol[i])) # Compute error at each iteration
        z=np.polyfit(ek[0:ek.shape[0]-1],ek[1:ek.shape[0]],1) # Fit line thru (x,y)->(log|e(k)|,log|e(k+
        print('Order of convergence',z[0])
        print('Iterations=',iterations)
        return xnew

def Secant(f,x1,x0):
        sol=np.zeros(100)#Create array of computed solutions
        xold=x1
        xoldest=x0
        sol[0]=xoldest
        sol[1]=xold
        iterations=2
        xnew=xold-(f(xold)/(f(xold)-f(xoldest)))/(xold-xoldest)# Compute first iteration
        sol[iterations]=xnew
        while(math.fabs(xnew-xold)>threshold and iterations<maxIterations):
                xoldest=xold
                xold=xnew
                xnew=xold-f(xold)/((f(xold)-f(xoldest))/(xold-xoldest))
                iterations+=1
                sol[iterations]=xnew

        ek=np.zeros(iterations);
        for i in range(iterations):
                ek[i]=math.log10(math.fabs(xnew-sol[i]))# Compute error at each iteration
        z=np.polyfit(ek[0:ek.shape[0]-1],ek[1:ek.shape[0]],1)# Fit line thru (x,y)->(log|e(k)|,log|e(k+1)|)
        print('Order of convergence',z[0])
        print('Iterations=',iterations)
        return xnew

def bisect(f,low,high):
        iterations=0;
        sol=np.zeros(100)#Create array of computed solutions
        while(low<=high and iterations<maxIterations):
                mid=(low*(1.)+high*(1.))/2. # Compute midpoint of interval
                sol[iterations]=mid
                if math.fabs(f(mid))<threshold:
                        low=high+1
                elif f(mid)*f(low)<0:
                        high=mid
                else:
                        low=mid
                iterations+=1

        ek=np.zeros(iterations);
        for i in range(iterations):
                if not math.fabs(mid-sol[i])==0:
                        ek[i]=math.log10(math.fabs(mid-sol[i]))# Compute error at each iteration
                else:
                        ek[i]=threshold
        z=np.polyfit(ek[0:ek.shape[0]-1],ek[1:ek.shape[0]],1)# Fit line through (x,y)->(log|e(k)|,log|e(k+1
        print('Order of convergence',z[0])
        print('Iterations=',iterations)
        return mid;

print("Newton's method on problem 1")
sol=Newton(f1,f1prime,1)
print('x=',sol)

print("Secant method on problem 1")
sol=Secant(f1,1.1,1.)
print('x=',sol)

print("Bisection method on problem 1")
sol=bisect(f1,0,5)
print('x=',sol)
```

The termination criterion that was used here was checking the absolute difference of x is greater than epsilon (threshold or not).

Problem 4

Recall the secant update to be $x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$

$x_k = x^* \Rightarrow x_{k+1} = x^* - \frac{f(x^*)(x^* - x_{k-1})}{f(x^*) - f(x_{k-1})} = x^* - \frac{0(x^* - x_{k-1})}{0 - f(x_{k-1})} = x^*$

$x_{k-1} = x^* \Rightarrow x_{k+1} = x_k - \frac{f(x_k)(x_k - x^*)}{f(x_k) - f(x^*)} = x_k - \frac{f(x_k)(x_k - x^*)}{f(x_k)} = x^*$