Homework 4 Solutions

November 27, 2017

1 Problem 1 Solution

1.1 Part a

By combining $\hat{x}_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$ into $x_{k+1} = \frac{x_k + \hat{x}_{k+1}}{2}$, we obtain

$$x_{k+1} = \frac{1}{2}\left(x_k + x_k - \frac{f(x_k)}{f'(x_k)}\right) = x_k - \frac{f(x_k)}{2f'(x_k)} = g(x_k)$$

Suppose this equation converges to a value $x^\star$. Then,

$$x^\star = g(x^\star) = x^\star - \frac{f(x^\star)}{2f'(x^\star)} \Rightarrow -\frac{f(x^\star)}{2f'(x^\star)} = 0 \Rightarrow f(x^\star) = 0$$

Thus, the iteration converges to a solution of $f(x) = 0$.

1.2 Part b

From above, we have $g(x) = x - \frac{1}{2}\frac{f(x)}{f'(x)}$, and $g'(x) = \frac{1}{2} + \frac{1}{2}\frac{f(x)f''(x)}{[f'(x)]^2}$. Assuming the same conditions used to prove that Newton's method converges ($f(a) = 0$, $f'(a) \neq 0$, and $f''(x)$ is bounded near a) gives

$$\lim_{x \to a} g'(x) = \frac{1}{2} + \frac{f(a)f''(a)}{2[f'(a)]^2} = \frac{1}{2}.$$

Since $|g'(a)| = 1/2 < 1$, it is a contraction, and thus, guaranteed to converge.

1.3 Part c

With the Taylor series approximation for $k = 1$, we obtain:

$$g(y) = g(x) + g'(c)(y - x)$$

for some number $c \in (x, y)$. With the equations from part b), we obtain:

$$\begin{aligned} g(x_k) &= g(a) + g'(a)(x_k - a) \\ \Rightarrow x_{k+1} &= a + \frac{1}{2}(x_k - a) \\ \Rightarrow |x_{k+1} - a| &= |\frac{1}{2}(x_k - a)| \\ \Rightarrow |e_{k+1}| &\leq \frac{1}{2}|(e_k)| \end{aligned}$$

The result has $d = 1$ in $|e_{k+1}| \leq L|e_k|^d$. By the definition in section 5.3 Order of Convergence, if $d = 1$, we shall also require that $L < 1$ in order to guarantee that the error e_k is being reduced. This requirement is satisfied as proved in part b) that $L = \frac{1}{2} < 1$, so this method exhibits linear convergence and the order of convergence is 1.

1.4 Breakdown of Points

The Points were broken down as follows:

Part a : 8 points
Part b : 6 points
Part c : 6 points

1.5 Common Mistakes

1.5.1 Used Examples

Some students used a specific example (i.e. $f(x) = x^2 - 28$) or graph to derive the conclusion without using general equations to prove.

1.5.2 Same as Newton's Method

Some students treated the whole proof the same as proving Newton's method, i.e., getting $\lim_{x \to a} g'(x) = 0$. And used the order of convergence for Newton's method for part c).

1.5.3 Using $k = 2$ for Taylor's Series Approximation

Some students used $k = 2$ in Taylor's theorem and derived the order of convergence as 2 with quadratic convergence.

2 Problem 2 Solutions

1. For the equation $f(x) = x^2 - 3x + 2 = 0$, we are given a series of equations such that $x = g_i(x)$ is equivalent to $f(x) = 0$. If we can show that $|g_i'(2)| < 1$, then g is a contraction on the interval $(2 - \delta, 2 + \delta)$ and the

fixed point iteration scheme is guaranteed to converge. Each of the $g_i'(x)$ are as follows:

- $g_1(x) = (x^2+2)/3 = (x^2/3) + (2/3) \Rightarrow g_1'(x) = 2x/3 \Rightarrow g_1'(2) = 4/3$
  Since $|g_1'(2)| = 4/3 > 1$, we have a divergent scheme.
- $g_2(x) = \sqrt{3x-2} \Rightarrow g_2'(x) = \frac{3}{2\sqrt{3x-2}} \Rightarrow |g_2'(2)| = \frac{3}{4} < 1$
  Since $3/4 < 1$, we have a convergent scheme.
- $g_3(x) = 3 - \frac{2}{x} \Rightarrow g_3'(x) = \frac{2}{x^2} \Rightarrow g_3'(2) = \frac{1}{2}$
  Since $|g_3'(2)| = 1/2 < 1$, we have a convergent scheme.
- $g_4(x) = \frac{x^2-2}{2x-3} \Rightarrow g_4'(x) = \frac{2x^2-6x+4}{(2x-3)^2} \Rightarrow g_4'(2) = 0$
  Since $|g_4'(2)| = 0 < 1$, we have a convergent scheme.

2. The Python code to confirm the convergence/divergence is given below:

2.1 Code for Bisection, Newton's and Secant Method

```
import numpy as np
import matplotlib.pyplot as plt
from decimal import *

def g1(x):
        return np.float64(((x**2)+2)/3)

def g2(x):
        return np.float64(np.sqrt(3*x-2))

def g3(x):
        return np.float64(3-(2/x))

def g4(x):
        return np.float64(((x **2)-2)/((x*2)-3))

def fxPointIteration(x0, f, tolerance, maxIter):
        X = []
        Y = []
        x1 = f(x0)
        count = []
        iterat = 0
        count.append(iterat)
        # X.append(np.log(np.float64(abs(x-2.0))))
        # Y.append(np.log(np.float64(abs(x1-2.0))))

        while(iterat < maxIter):
                iterat = iterat+1
                x0 = x1
```

```
                x1 = f(x1)
                # print x0
                # print x1
        print np.float64(abs(x1-2.0))
        if ((np.float64(abs(x1-2.0)))<tolerance):
                print "We are converging towards the
                    solution"
        else:
                print "We have not converged"
                # print x1
                count.append(iterat)


        return x1


print "NOW WE WILL SEE THE CONVERGENCE BETWEEN THE
    ITERATION SCHEMES"
print "~~~~~~~~~~~~~~~~~~~~~~~~~~G1
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

x0 = 1
maxIter = 100
tolerance = 10^-12

x1 = fxPointIteration(x0, g1, tolerance, maxIter)

print "~~~~~~~~~~~~~~~~~~~~~~~~~~G2
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

x1 = fxPointIteration(x0, g1, tolerance, maxIter)

print "~~~~~~~~~~~~~~~~~~~~~~~~~~G3
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

x1 = fxPointIteration(x0, g1, tolerance, maxIter)

print "~~~~~~~~~~~~~~~~~~~~~~~~~~G4
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

x1 = fxPointIteration(x0, g1, tolerance, maxIter)
```

The results from the above code are as expected. The scheme $g_1(x)$ does not converge to the exact solution of 2. The other functions, $g_2(x)$, $g_3(x)$, and $g_4(x)$ converge to 2.

3 Problem 3 Solution

3.1 Code for Bisection, Newton's and Secant Method

```
from sympy import *
x = symbols("x")

def bisection(expr, a, b, epsilon, nmax):
    n = 0
    while (n < nmax):
        c = (a + b) / 2.0
        if ((expr.subs(x,c)==0) or ((b-a)/2<epsilon)):
            print("Bisection Method: %.8f" % c)
            return
        n = n + 1
        if ((expr.subs(x, c) > 0) and (expr.subs(x, a) > 0)
            or
            (expr.subs(x, c) < 0) and (expr.subs(x, a) < 0))
                :
            a = c
        else:
            b = c
    print("Method failed.")

def newtons(expr, init_guess, epsilon, nmax):
    n = 0
    x_prev = init_guess
    x_curr = 0
    while (n < nmax):
        x_curr = x_prev - (expr.subs(x, x_prev) / diff(expr,
             x).subs(x, x_prev).evalf())
        if (abs(x_curr - x_prev) < epsilon):
            print("Newton's Method: %.8f" % x_curr)
            return
        x_prev = x_curr
        n = n + 1
    print("Method failed.")

def secant(expr, x_0, x_1, epsilon, nmax):
    n = 0
    while (n < nmax):
        x_2 = (x_0 * expr.subs(x, x_1).evalf() - x_1 * expr.
            subs(x, x_0).evalf()) / \
              (expr.subs(x, x_1).evalf() - expr.subs(x, x_0)
                  .evalf())
        if (abs(x_2 - x_1) < epsilon):
            print("Secant Method: %.8f" % x_2)
            return
```

```
        x_0 = x_1
        x_1 = x_2
        n = n + 1
    print("Method failed.")

def test_methods(expr, x_0, x_1, epsilon, nmax):
    newtons(expr, x_0, epsilon, nmax)
    secant(expr, x_0, x_1, epsilon, nmax)
    bisection(expr, x_0, x_1, epsilon, nmax)

expr = x**3 - 2*x - 5
test_methods(expr, 1, 5, 0.000000001, 1000)

expr = exp(-x) - x
test_methods(expr, -1, 3, 0.000000001, 1000)

expr = x * sin(x) - 1
test_methods(expr, 1, 2, 0.000000001, 1000)

expr = x**3 - 3*x**2 + 3*x - 1
test_methods(expr, 0, 3, 0.000000001, 1000)
```

3.2 Order of Convergence

For determining the order of convergence, we are interested in the following equation:

$$|e_{k+1}| = C|e_k|^\alpha$$

Taking the log of both sides and simplifying we get:

$$\log|e_k + 1| = \log(C) + \alpha \log(|e_k|)$$

We note that this equation has the form:

$$y = mx + c$$

The only unknown in the above equation is α. Therefore, to determine the order of convergence, all that is necessary is to update the three methods defined above for bookkeeping on these variables for each iteration. Then we can determine the slope, and therefore α, by linear regression.

3.3 Common Mistakes

The only mistake for the majority on this problem was not determining the order of convergence.

4 Problem 4 Solution

With the secant method, we can obatin that

$$x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}}$$

for the k^{th} iteration for solving a nonlinear equation. If $x_k = x^\star$, then $f(x_k) = 0$. Therefore,

$$x_{k+1} = x_k - \frac{f(x_k)}{\frac{f(x_k)-f(x_{k-1})}{x_k - x_{k-1}}} = x_k - \frac{(x_k - x_{k-1})\cdot 0}{0 - f(x_{k-1})} = x_k = x^\star.$$

Else if $x_{k-1} = x^\star$, then $f(x_{k-1}) = 0$. Therefore,

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

However, not both x_k and x_{k-1} can be equal to $x^\star$, because this will lead to undefined situation within previous steps.

4.1 Common Mistakes

4.1.1 Missing $x_{k-1} = x^\star$ situation

Both situations of $x_k = x^\star$ and $x_{k-1} = x^\star$ need to be considered, but some students only proved that $x_k = x^\star \Rightarrow x_{k+1} = x^\star$.