Applied Mathematics Assignment 02

Bhekimpilo Ndhlela (18998712)

02 March 2018

Question 1

Python Source Code:

```
#!/usr/bin/python
def fpi(f, x, k):
    for i in xrange(k):
        print i + 1, "\t{:.15f}".format(x)
        x = f(x)
    print "\n"

if __name__ == "__main__":
    x, k = 1, 10
    f = lambda x : (0.5 * x) + (1 / x)
    g = lambda y : ((2. * x) / 3.) + (2. / (3. * x))
    h = lambda z : (0.75 * x) + (0.5 * x)
    print "The itterations for F(x):"
    fpi(f, x, k)
    print "The itterations for G(x):"
    fpi(g, x, k)
    print "The itterations for H(x):"
    fpi(h, x, k)
```

i	a.) F(x)	b.) G(x)	c.) H(x)
1	1.000000000000000	1.000000000000000	1.000000000000000
2	1.500000000000000	1.333333333333333	1.250000000000000
3	1.416666666666667	1.333333333333333	1.250000000000000
4	1.414215686274510	1.333333333333333	1.250000000000000
5	1.414213562374690	1.333333333333333	1.250000000000000
6	1.414213562373095	1.333333333333333	1.250000000000000
7	1.414213562373095	1.333333333333333	1.250000000000000
8	1.414213562373095	1.333333333333333	1.250000000000000
9	1.414213562373095	1.333333333333333	1.250000000000000
10	1.414213562373095	1.333333333333333	1.250000000000000

Question 2

a.)

Python Source Code:

```
#!/usr/bin/python
def sign(x):
    if x < 0:   return -1
    elif x > 0: return 1
    else:            return 0

def bisect(f, a, b, tol):
    fa, fb = f(a), f(b)

    # assumming f(a)f(b)<0 is satisfied!
    while (b-a)/2. > tol:
        c  = (a + b) / 2.0
        fc = f(c)
        if fc == 0:                   # c is a solution, done
            return c
        elif sign(fc)*sign(fa) < 0:   # a and c make the new interval
            b, fb = c, fc
        else:
            a, fa = c, fc
    return (a+b)/2.                   # new midpoint is best estimate

if __name__ == "__main__":
    import sys
    from math import (cos, tan)
    f = lambda theta: 20.0 * tan(theta) - ((20.0**2 * 9.81) / (2 * (17.0**2)
    print "{:.15f}".format(bisect(f,0,1, 1.0e-5))
```

Answer: 0.554908752441406

Question 3

Python Source Code for question 3 a.), b.) and c.):

```
def question_3a(f, debug=True):
    x = [.9, 0, 0, 0, 0, 0]
    for i in xrange(5):
        fx, dx = f(x[i])
        x[i+1] = x[i] - (fx / dx)

    if debug is True:
        print "Question 3 (a) results:"
        for i in xrange(len(x)):
            print i + 1, "\t", "{:.15f}".format(x[i])

def question_3b(g, debug=True):
    x = [.9, 0, 0, 0, 0, 0]
    for i in xrange(5):
        gx, dx = g(x[i])
        x[i+1] = x[i] - (gx / dx)

    if debug is True:
        print "Question 3 (b) results:"
        for i in xrange(0 ,len(x)):
            print i + 1, "\t", "{:.15f}".format(x[i])

def question_3d(g, debug=True):
    x = [.9, 0, 0, 0, 0, 0]
    # m == multiplicity of the function g(x)
    m = 2
    for i in xrange(5):
        gx, dx = g(x[i])
        x[i+1] = x[i] - (m * (gx / dx))

    if debug is True:
        print "Question 3 (d) results:"
        for i in xrange(0 ,len(x)):
            print i + 1, "\t", "{:.15f}".format(x[i])

from numpy import exp
f = lambda x: (x * exp(x - 1) - 1, exp(x-1) + x * exp(x-1))
g = lambda x: (-x * exp(1 - x) + 1, -exp(1-x) + x * exp(1-x))

question_3a(f)
question_3b(g)
question_3d(g)
```

a.)

$x_0 = 0.900000000000000$

i	**Secant Method**
1	0.900000000000000
2	1.007984693724025
3	1.000047584190120
4	1.000000001698142
5	1.000000000000000
6	1.000000000000000

b.)

$x_0 = 0.900000000000000$

i	**Secant Method**
1	0.900000000000000
2	0.948374180359597
3	0.973748560382854
4	0.986760173690201
5	0.993350967791509
6	0.996668127855918

c.)

$$f(x) = xe^{(x-1)} - 1$$
$$f(r) = f(1) = 0$$
$$f^{`}(x) = e^{(x-1)} + xe^{(x-1)}$$
$$f^{`}(r) = f^{`}(1) = 2 \neq 0$$
$$multiplicity = m = 1$$

$$g(x) = -xe^{(1-x)} + 1$$
$$g(r) = g(1) = 0$$
$$g^{`}(x) = -e^{(1-x)} + xe^{(1-x)}$$
$$g^{`}(r) = g^{`}(1) = 0$$
$$g^{``}(x) = e^{(1-x)} - (x-1)e^{(1-x)} = -(x-2)e^{(1-x)}$$
$$g^{``}(r) = g^{``}(1) = 1 \neq 0$$
$$multiplicity = m = 2$$

Since: $m = 1$**, for** $f(x)$**, then** $f(x)$ **is well conditioned.**
However, $m = 2$ **for** $g(x)$**, this implies that** $g(x)$ **is ill-conditioned.**

d.)

$x_0 = 0.900000000000000$
$multiplicity = m = 2$

i	Secant Method
1	0.900000000000000
2	0.996748360719194
3	0.999996478477208
4	1.000000000070078
5	1.000000000070078
6	1.000000000070078

Since: $m = 2$ **for** $g(x)$**, this implies that** $g(x)$ **is ill-conditioned, and hence the root has not been computed to full precision with this method (This is because of rounding off errors). The ill-conditioning has not been thwarted.**

Question 4

Python Source Code:

```
def secant_method(x, debug=True):
    appr_list = [1., 0.5, 0., 0., 0., 0., 0., 0., 0.]
    for i in xrange(1, len(appr_list) - 1):
        num = appr_list[i] * appr_list[i - 1] + x
        den = appr_list[i] + appr_list[i - 1]
        appr_list[i + 1] = num / den

    if debug is True:
        print "The Secant Method, DEBUG MODE: ON:"
        for i in xrange(len(appr_list)):
            print i + 1, "\t", "{0:.15f}".format(appr_list[i])
    return appr_list

def newton_method(x, debug=True):
    appr_list = [1., 0., 0., 0., 0., 0., 0., 0., 0.]
    for i in xrange(len(appr_list) - 1):
        num = appr_list[i]**2 + x
        den = appr_list[i] * 2
        appr_list[i + 1] = num / den

    if debug is True:
        print "The Newton Method, DEBUG MODE: ON:"
        for i in xrange(len(appr_list)):
            print i + 1, "\t", "{0:.15f}".format(appr_list[i])
    return appr_list
import math
new_res, sec_res = newton_method(1. / 9.), secant_method(1. / 9.)
```

i	Secant Method	Newton's Method
1	1.000000000000000	1.000000000000000
2	0.500000000000000	0.555555555555556
3	0.407407407407407	0.377777777777778
4	0.346938775510204	0.335947712418301
5	0.334669338677355	0.333343506014598
6	0.333360001066709	0.333333333488554
7	0.333333386666671	0.333333333333333
8	0.333333333335467	0.333333333333333
9	0.333333333333333	0.333333333333333

Question 5

Python Source Code:

```
#!/usr/bin/python
def newton_method_sys(fxy, j0, j1, debug=True):
    xn = zeros((2, 9))      #store itteration results for x^[n + 1]
    jx = zeros((2, 2))      #store currant itteration jacobian inverse
    fx = zeros((2, 1))      #store the results of the f(x^[n]) for a particul
    sx = zeros((2, 1))

    for i in xrange(len(xn[1]) - 1):
        jx[0][0], jx[0][1] = j0(xn[0][i], xn[1][i])
        jx[1][0], jx[1][1] = j1(xn[0][i], xn[1][i])
        fx[0][0], fx[1][0] = fxy(xn[0][i], xn[1][i])
        sx = linalg.solve(negative(jx), fx)
        xn[0][i + 1] = xn[0][i] + sx[0]
        xn[1][i + 1] = xn[1][i] + sx[1]

    if debug is True:
        print "xn = |", xn[0][-1], xn[1][-1], "|"

if __name__ == "__main__":
    from numpy import (array, zeros, exp, linalg, negative)
    from math import (cos, sin)
    fxy = lambda x, y: (x * exp(y) + y - 7, sin(x) - cos(y))
    j0 = lambda x, y: (exp(y), x * exp(y) + 1)
    j1 = lambda x, y: (cos(x), sin(y))
    newton_method_sys(fxy, j0, j1)
else:
    import sys
    sys.exit("please run as client...")
```

when:
$x_0 = [0, 0]^T$
Then:
$x^* = 0.0199681607333$
And also:
$f(x^*) = 4.73235714112$