# Assignment 2

Doug Shi-Dong 260466662
MATH-578 Numerical Analysis

October 8, 2015

## Additional Problem 1

When $x_0$ is chosen to be -1, the starting point is relatively far from the solution. Additionally, the first derivative of approximately $-0.16$ causes a big change in the next iteration. Once the fixed point is in the neighbourhood of the solution, it starts converging quadratically.

| Iteration | $x_i$ | $\lvert f(x) \rvert$ |
|---|---|---|
| 0 | -1.0000000000000000 | 1.5403023058681398 |
| 1 | 8.7162169587795688 | 9.4755160367702089 |
| 2 | 2.9760606550969122 | 3.9623914888656024 |
| 3 | -0.4257846886852303 | 1.3364995977405378 |
| 4 | 1.8511838177311790 | 2.1279118458102908 |
| 5 | 0.7660395196449308 | 0.0453774576567640 |
| 6 | 0.7392410674960820 | 0.0002609824734986 |
| 7 | 0.7390851385832758 | 0.0000000089841422 |
| 8 | 0.7390851332151607 | 0.0000000000000000 |

Table 1: Fixed Point Iterations. Starting point $x_0 = -1$

When the starting point is in the neighbourhood of the solution, the solution converges very quickly. After 3 iterations, the computed solution has 16 digits of accuracy.

| Iteration | $x_i$ | $\lvert f(x) \rvert$ |
|---|---|---|
| 0 | 0.7853981633974483 | 0.0782913822109007 |
| 1 | 0.7395361335152383 | 0.0007548746825027 |
| 2 | 0.7390851781060102 | 0.0000000751298664 |
| 3 | 0.7390851332151611 | 0.0000000000000008 |

Table 2: Fixed Point Iterations. Starting point $x_0 = \pi/4$

# Additional Problem 2

The algorithm converges in 4 iterations.

| Iteration | $x_i$ | $\|f(x)\|$ |
|---|---|---|
| 0 | 1.000000 | 2.236068 |
| | 1.000000 | |
| 1 | 0.611111 | 0.428289 |
| | 0.833333 | |
| 2 | 0.503659 | 0.045328 |
| | 0.852494 | |
| 3 | 0.499964 | 0.000143 |
| | 0.866046 | |
| 4 | 0.500000 | 0.000000 |
| | 0.866025 | |

Table 3: Fixed Point Iterations. Starting point $x_0 = -1$

# Newton's Method 1D

```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt

fx = lambda x: np.cos(x) - x

dfdx = lambda x: -np.sin(x) - 1

maxit = 20
tol = 2e-15
x = np.empty(maxit)
f = np.empty(maxit)
# Starting Point x0
x[0] = np.pi/4

for i in range(maxit):
    f[i] = fx(x[i])
    if(abs(f[i]) < tol):
        break
    x[i+1] = x[i] - f[i] / dfdx(x[i])


for ii in range(i+1):
    print '%d & %.16f & %.16f \\\\' % (ii, x[ii], abs(f[ii]))
```

# Newton's Method 2D

```python
#!/usr/bin/env python
import numpy as np

def fun(x):
    y = np.array([
        3 * x[0]**2 - x[1]**2,
        3 * x[0] * x[1]**2 - x[0]**3 - 1,
    ])
    return y

def Jac(x):
    J  = np.empty((2,2))
    J[0][0] = 6 * x[0]
    J[0][1] = -2 * x[1]
    J[1][0] = 3 * x[1]**2 - 3 * x[0]**2
    J[1][1] = 6 * x[0] * x[1]
    return J


maxit = 10
tol = 1e-6
x = np.zeros(shape=(maxit,2), dtype=np.float64)
err = np.zeros(maxit, dtype=np.float64)
x[0] = [1.,1.]
print('x0:\n', x[0] )
for i in range(maxit):

    fx = fun(x[i,:])

    err[i] = np.linalg.norm(fx,2)
    if(err[i] < tol):
        break

    J = Jac(x[i,:])

    J = np.linalg.inv(J)

    x[i+1,:] = x[i,:] - np.dot(J, fx)


for ii in range(i+1):
    print '%d & %.6f & %.6f \\\\' % (ii, x[ii,0], err[ii])
    print '    & %.6f &         \\\\' % (x[ii,1])
```