Ting Fung Lam 2924629375

MATH 501 PA3

The program language is python

Question 1
After input for 4.5 it found the root is 4.4934. For 7.7 the root is at 7.7252

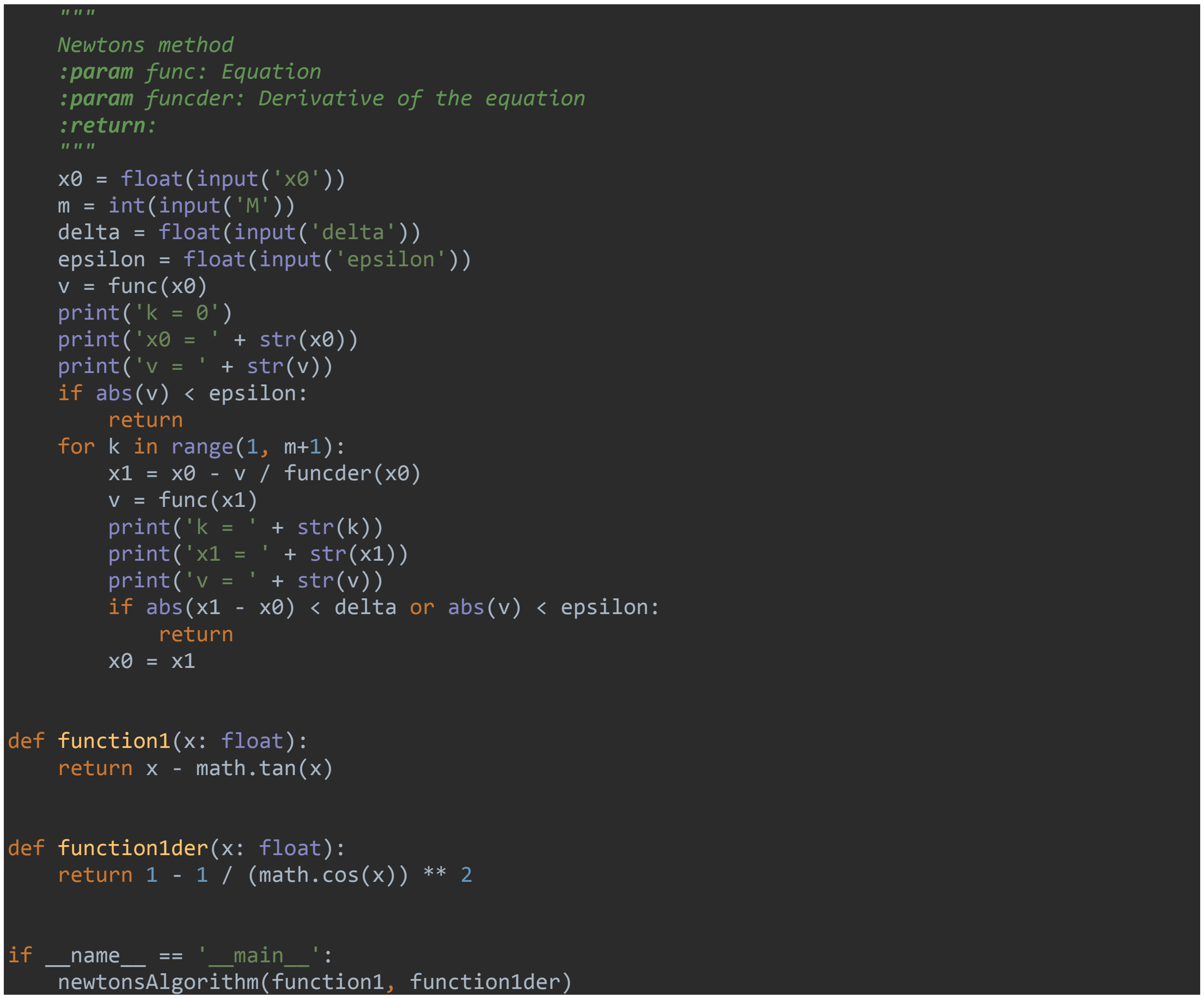
Output:

```
x04.5
M100
delta0.0001
epsilon0.0001
k = 0
x0 = 4.5
v = -0.1373320545511847
k = 1
x1 = 4.493613902743204
v = -0.004131873789261498
k = 2
x1 = 4.493409655013248
v = -3.979680768928517e-06

x07.7
M100
delta0.0001
epsilon0.0001
k = 0
x0 = 7.7
v = 1.2571275265074489
k = 1
x1 = 7.730284490072234
v = -0.3127020377423255
k = 2
x1 = 7.7254506082477565
v = -0.011881125113178292
k = 3
x1 = 7.72525214726643
v = -1.852031312932212e-05
```

```python
import math


def newtonsAlgorithm(func, funcder):
```



```
    """
    Newtons method
    :param func: Equation
    :param funcder: Derivative of the equation
    :return:
    """
    x0 = float(input('x0'))
    m = int(input('M'))
    delta = float(input('delta'))
    epsilon = float(input('epsilon'))
    v = func(x0)
    print('k = 0')
    print('x0 = ' + str(x0))
    print('v = ' + str(v))
    if abs(v) < epsilon:
        return
    for k in range(1, m+1):
        x1 = x0 - v / funcder(x0)
        v = func(x1)
        print('k = ' + str(k))
        print('x1 = ' + str(x1))
        print('v = ' + str(v))
        if abs(x1 - x0) < delta or abs(v) < epsilon:
            return
        x0 = x1


def function1(x: float):
    return x - math.tan(x)


def function1der(x: float):
    return 1 - 1 / (math.cos(x)) ** 2


if __name__ == '__main__':
    newtonsAlgorithm(function1, function1der)
```

Question 34
The solution is x = 0.1342 y = 1.3041

```
x01
y01
M100
delta0.01
epsilon0.01
k = 0
x0 = [1. 1.]
v = [ 41. 263.]
k = 1
x1 = [0.40671909 0.1542173 ]
v = [ 2.86139352 61.63104438]
k = 2
x1 = [0.21006854 1.56132523]
v = [ 7.91981091 12.47533097]
```

```
k = 3
x1 = [0.14061668 1.30006648]
v = [0.27302453 1.01995028]
k = 4
x1 = [0.13423928 1.3041358 ]
v = [6.62373383e-05 6.92311811e-03]
```

```python
import numpy as np


def newtonNonLinear(f, fdx, fdy):
    """
    Non-linear system newtons method
    :param f: List of equations
    :param fdx: List of partial derivative x of the equations
    :param fdy: List of partial derivative y of the equations
    :return:
    """
    # Input for vector x0
    x00 = float(input('x0'))
    x10 = float(input('y0'))
    x0 = np.array([x00, x10])

    m = int(input('M'))
    delta = float(input('delta'))
    epsilon = float(input('epsilon'))
    v = np.array([func(x0) for func in f])
    print('k = 0')
    print('x0 = ' + str(x0))
    print('v = ' + str(v))

    if abs(np.max(v)) < epsilon:
        return
    for k in range(1, m+1):
        J = np.array([[funcx(x0), funcy(x0)] for (funcx, funcy) in (fdx, fdy)]).T
        x1 = x0 - np.dot(np.linalg.inv(J), v)
        v = np.array([func(x1) for func in f])
        print('k = ' + str(k))
        print('x1 = ' + str(x1))
        print('v = ' + str(v))
        if abs(np.max(x1 - x0)) < delta or abs(np.max(v)) < epsilon:
            return
        x0 = x1


def f1(x):
    return 4 * x[1] ** 2 + 4 * x[1] + 52 * x[0] - 19


def f1dx(x):
    return 52


def f1dy(x):
    return 8 * x[1] + 4


def f2(x):
    return 169 * x[0] ** 2 + 3 * x[1] ** 2 + 111 * x[0] - 10 * x[1] - 10


def f2dx(x):
    return 338 * x[0] + 111


def f2dy(x):
    return 6 * x[1] - 10


if __name__ == '__main__':
    f = [f1, f2]
    fdx = [f1dx, f2dx]
    fdy = [f1dy, f2dy]
    newtonNonLinear(f, fdx, fdy)
```