1)

(a) Write the equation for the tangent line to $y = f(x)$ at $x = p$.

→ The equation of a line passing through $p, f(p)$ with slope m is given as

$$y - f(p) = m(x - p)$$

Since the line is tangent to the curve at $x = p$, the slope $m = f'(p)$. Hence,

$$y - f(p) = f'(p)(x - p) \Rightarrow y = f'(p)x + [f(p) - pf'(p)]$$

(b) Solve for the x-intercept of the line in equation (a).

→ The x-intercept is defined as the value of x such that $y = 0$. We see that

$$0 = f'(p)x + [f(p) - pf'(p)] \Rightarrow x = \frac{pf'(p) - f(p)}{f'(p)} = p - \frac{f(p)}{f'(p)}$$

assuming that $f'(p) \neq 0$.

(c) Write the equation for the line that intersects the curve $y = f(x)$ at $x = p$ and $x = q$.

→ The slope of the line passing through $(p, f(p))$ and $(q, f(q))$ is

$$m = \frac{f(p) - f(q)}{p - q}$$

so we see

$$y - f(p) = \frac{f(p) - f(q)}{p - q}(x - p) \Rightarrow y = \frac{f(p) - f(q)}{p - q}(x - p) + f(p)$$

(d) Solve for the x-intercept of the line in equation (c).

→ We can solve for the x-intercept for the equation in part (c) as we did for that in part (b):

$$\begin{aligned} 0 &= \frac{f(p) - f(q)}{p - q}(x - p) + f(p) \\ \frac{f(p) - f(q)}{p - q}(x - p) &= -f(p) \\ x - p &= -f(p)\frac{p - q}{f(p) - f(q)} \\ x &= p - \frac{f(p)(p - q)}{f(p) - f(q)} \end{aligned}$$

2) Starting with (0,1), perform two iterations of Newton iteration on the following systems of nonlinear equations

$$\begin{cases} 4x_1^2 - x_2^2 = 0 \\ 4x_1x_2^2 - x_1 = 1 \end{cases}$$

It has been shown that for the system

$$\begin{cases} f_1(x_1, x_2) = 0 \\ f_2(x_1, x_2) = 0 \end{cases}$$

we can find successive approximations to the root as follows:

$$\vec{x}_{n+1} = \vec{x}_n - J^{-1} f(\vec{x}_n)$$

where

$$\vec{x}_n = \begin{pmatrix} x_1^{(n)} \\ x_2^{(n)} \end{pmatrix} \quad f(\vec{x}_n) = \begin{pmatrix} f_1(x_1^{(n)}, x_2^{(n)}) \\ f_2(x_1^{(n)}, x_2^{(n)}) \end{pmatrix}$$
$$J(x_1, x_2) = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{pmatrix}$$

For our problem $f_1 = 4x_1^2 - x_2^2$ and $f_2 = 4x_1 x_2^2 - x_1 - 1$, so Newton iteration gives

$$\begin{pmatrix} x_1^{(n+1)} \\ x_2^{(n+1)} \end{pmatrix} = \begin{pmatrix} x_1^{(n)} \\ x_2^{(n)} \end{pmatrix} - \begin{pmatrix} 8x_1^{(n)} & -2x_2^{(n)} \\ 4(x_2^{(n)})^2 - 1 & 8x_1^{(n)} x_2^{(n)} \end{pmatrix}^{-1} \begin{pmatrix} 4(x_1^{(n)})^2 - (x_2^{(n)})^2 \\ 4x_1^{(n)}(x_2^{(n)})^2 - x_1^{(n)} - 1 \end{pmatrix}$$

If we begin with $x_1^{(0)} = 0$ and $x_2^{(0)} = 1$, then

$$\begin{pmatrix} x_1^{(1)} \\ x_2^{(1)} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & -2 \\ 3 & 0 \end{pmatrix}^{-1} \begin{pmatrix} -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} - \begin{pmatrix} -1/3 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/2 \end{pmatrix}$$

Repeating for a second time we see

$$\begin{pmatrix} x_1^{(2)} \\ x_2^{(2)} \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/2 \end{pmatrix} - \begin{pmatrix} 8/3 & -1 \\ 0 & 4/3 \end{pmatrix}^{-1} \begin{pmatrix} 7/36 \\ -1 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/2 \end{pmatrix} - \begin{pmatrix} -5/24 \\ -3/4 \end{pmatrix} = \begin{pmatrix} 13/24 \\ 5/4 \end{pmatrix}$$

3) Write a program to compute the square root of m by bisection method when $m = 7,\ 13,\ 17$. The interval $[a, b]$ where the square root is located is $[s, s+1]$, where s is an integer and $s^2 < m < (s+1)^2$. Please find the approximate solution with 7 iteration steps, i.e. find $c_0, c_1, \ldots, c_7$.

→ The bisection method program is shown below and gives the following results after 7 iteration steps.

m	s	$\sqrt{m}$
7	2	2.644531
13	3	3.605469
17	4	4.121094

An image of the program output is displayed below to demonstrate the successive approximations:

```
sqrt(7) = 2.644531
iterations: [2.5, 2.75, 2.625, 2.6875, 2.65625, 2.640625, 2.6484375, 2.64453125]

sqrt(13) = 3.605469
iterations: [3.5, 3.75, 3.625, 3.5625, 3.59375, 3.609375, 3.6015625, 3.60546875]

sqrt(17) = 4.121094
iterations: [4.5, 4.25, 4.125, 4.0625, 4.09375, 4.109375, 4.1171875, 4.12109375]
```

```python
#!/usr/bin/env python3

import numpy as np
import matplotlib.pyplot as plt

def bisection(f,a,b,M=1e5,delta=1e-10,eps=1e-10):
    u = f(a)
    v = f(b)
    C = []

    if u*v > 0:
        c = 'Error: f(a) and f(b) have same sign'
    elif u == 0.0:
        c = a
    elif v == 0.0:
        c = b
    else:
        c = (a+b)/2
        m = f(c)
        for i in range(int(M)):
            C.append(c)
            if abs(a-b) < delta or abs(m) < eps:
                break
            if f(a)*f(c) < 0:
                b = c
                v = m
            else:
                a = c
                u = m
            c = (a+b)/2
            m = f(c)
    C.append(c)
    return c,C

m = [7,13,17]
s = 2.0
for i in range(len(m)):
    f = lambda x: x**2-m[i]
    a = s
    b = s+1
    root,iterations = bisection(f,a,b,M=7)
    print('sqrt(%d) = %f'%(m[i],root))
    print('iterations:',iterations)
    print()
    s += 1
```

4) Find the root of $f(x) = x^2 - 3$ by Newton's method, with initial guess $x_0 = -100, -5,$

$-0.001, 0, 0.001, 5, 100$. Discuss your findings.

→ The results of the program which computes the answers for each initial guess using Newton's method is shown below and the results are shown in the table below and in the figure of the program output below. It is seen that the results are similar in magnitude. Notice that when the initial guess and the root have the same sign, which is expected since the function is convex if restrict our domain to either $\mathbb{R}^-$ or $\mathbb{R}^+$ with a range $\mathbb{R}^+$, which guarantees that the algorithm converges to the simple root on the domain. Trivially, it is also observed that an initial guess of $x_0 = 0$ gives an error since $f'(0) = 0$

x_0	root
-100	-1.73205
-5	-1.73205
-0.001	-1.73205
0	Error
0.001	1.73205
5	1.73205
100	1.73205

```
start = -100.0 --> root = -1.7320508075688787

start = -5.0 --> root = -1.7320508075688774

start = -0.001 --> root = -1.7320508075688774

start = 0.0 --> root = Error: horizontal tangent reached

start = 0.001 --> root = 1.7320508075688774

start = 5 --> root = 1.7320508075688774

start = 100 --> root = 1.7320508075688787
```

```python
#!/usr/bin/env python3

import numpy as np
import sympy as sp

def newton(f,fp,r,M=1e5,delta=1e-10,eps=1e-10):
    u = f(r)
    up = fp(r)
    for i in range(int(M)):
        if abs(u) < eps:
            break
        if up == 0.0:
            r = 'Error: horizontal tangent reached'
            break
        r1 = r - u/up
        if abs(r1-r) < delta:
            r = r1
            break
        r = r1
        u = f(r)
        up = fp(r)
```

```
    return r

x = sp.symbols('x',real=True)
f_sym  = x**2 - 3
fp_sym = sp.diff(f_sym,x)
f  = sp.lambdify((x),f_sym,'numpy')
fp = sp.lambdify((x),fp_sym,'numpy')

r = [-100.0,-5.0,-0.001,0.0,0.001,5,100]
for val in r:
    print('start = {} --> root = {}'.format(val,newton(f,fp,val)))
    print()
```

5) Write down the first three iterations of the secant method for solving $x^2 - 3 = 0$, starting with $x_0 = 0$ and $x_1 = 1$. Meanwhile, programming with MATLAB to obtain approximation accurate within 10^{-6}.

The secant method follows an iterative algorithm defined by the recursion relation

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n+1})}$$

Thus, if $f(x) = x^2 - 3$, $x_0 = 0$, and $x_1 = 1$, then

$$\begin{aligned} x_2 &= 1 - (-2)\frac{1-0}{-2-(-3)} = 3 \\ x_3 &= 3 - (6)\frac{3-1}{6-(-2)} = \frac{3}{2} \\ x_4 &= \frac{3}{2} - \left(-\frac{3}{4}\right)\frac{3/2-3}{-3/4-6} = \frac{5}{3} = 1.\bar{6} \end{aligned}$$

Using a computer to obtain an approximation accurate to the sixth decimal place: 1.732051. Below are the figure showing the output of the program and the code used to implement the secant method.

```
x0 = 0.0, x1 = 1.0, root = 1.732051
```

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

import numpy as np

def secant(f,x0,x1,M=1e5,delta=1e-10,eps=1e-10):
    u = f(x0)
    v = f(x1)
    for i in range(int(M)):
        if abs(x1-x0) < delta:
            break
        if abs(u) < eps:
            x1 = x0
            break
        elif abs(v) < eps:
            break
        temp = x1
        x1 = x1 - v*(x1-x0)/(v-u)
```

```
            x0 = temp
            u = v
            v = f(x1)
        return x1

f = lambda x: x**2 - 3
x0 = 0.0
x1 = 1.0
print('x0 = {}, x1 = {}, root = {:.6f}'.format(x0,x1,secant(f,x0,x1,delta=1
    e-6)))
```