FMN050 - Assignment 1

Hugo Hjertén elt12hhj, Axel Lundholm elt12alu, Adnan Mehmedagic elt12ame, Albin Johansson elt12ajo

Task 1

Plotting the function was relatively easy, with only minor problems importing the correct packages. For the bisection method itself, an interval and a function was sent into the method. The method then decided whether or not these met the criteria (to have only one root within the interval), by looking at the product f(start)*f(stop). If this product wasn't negative an exception was raised. Otherwise the method split the interval in two and decided the correct new interval (containing the root). When the interval was smaller than the tolerance the method stopped and returned the approximate value.

Task 2

For us to have a reference we also implemented root finding with fsolve.

Task 3

The iteration method can most definitely be used to find the root of $f(x) = x^2 - 2$! With a very few iterations. With a starting point at x = 2 only 4 iterations were needed to find a root smaller than the tolerance. Since only one point is needed to find the next iteration this is a 1-point iteration. It is also non-encompassing and there is no need to calculate a derivative. Therefore this method is a fixed-point iteration.

Task 4

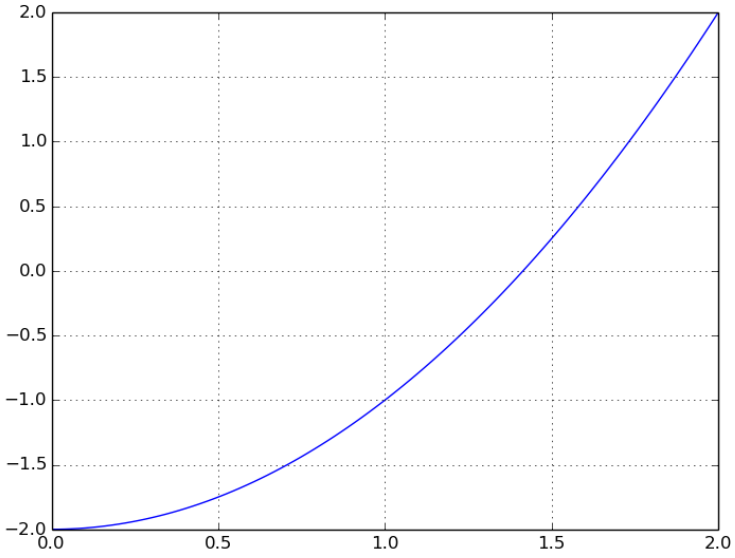
Newton was implemented by calculating the derivative for the function in the specific point. A test was then made, in order to catch possible by-zero-divisions. The value was calculated iteratively and the iterations stopped when the change in each iteration was lesser than the given tolerance. Protection for 'timeouts' was also implemented. For the secant method (which is a two point method), a line is 'drawn' between the two points. The x-value where the line meets the x-axis is then passed on to the next iteration. The iterations stop when either the interval is lesser than the tolerance or the function value is lesser than the tolerance.

Task 5

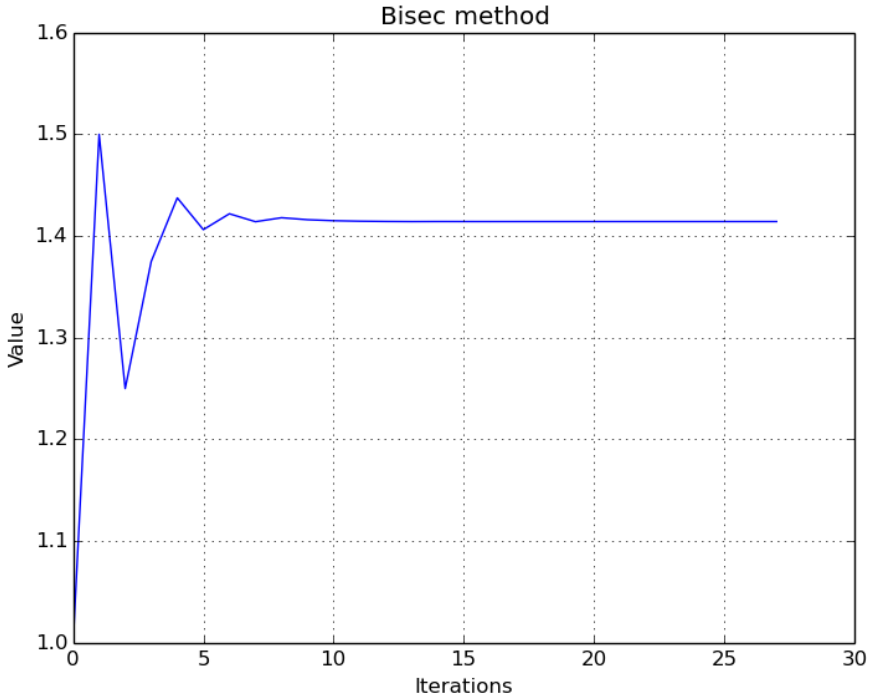
By plotting all the iterations (see figures below) we can how long it takes for the different methods to reach the root with desired precision, for the function $f(x) = x^2 - 2$ with relevant iteration start point values. In the first figure for each method we also see how much the value deviates from the desired final root of sqrt of 2 (1.41421356237). In the second figure for each method we see how much the iteration has changed in value from the previous iteration.

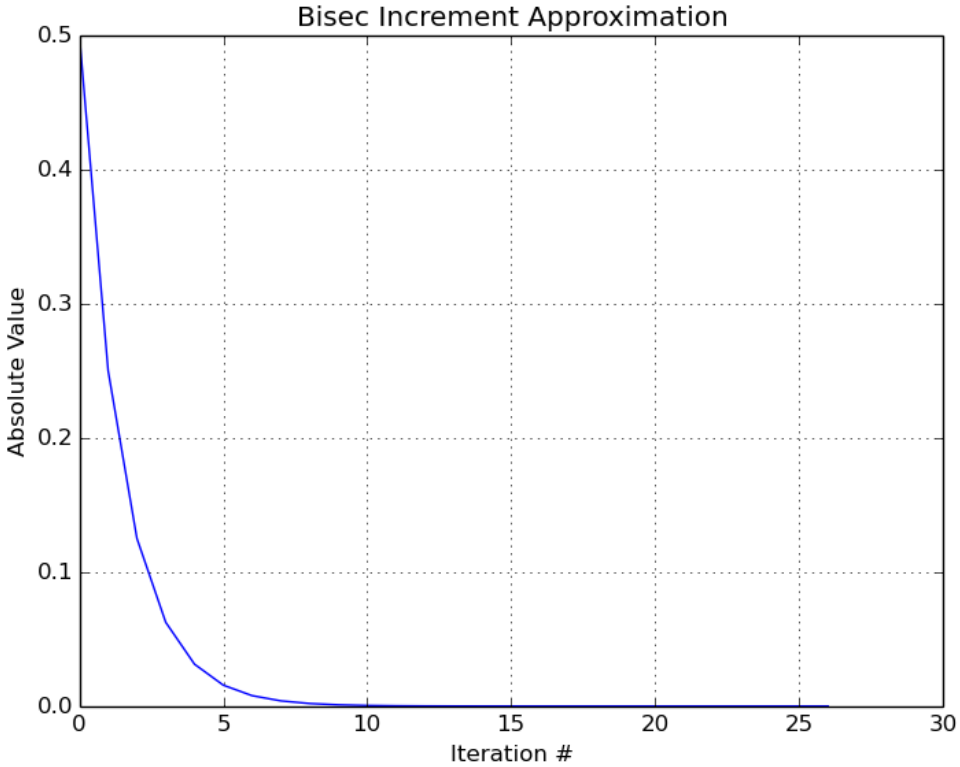
The fastest iteration seems to have been the unknown iteration (which should be a fixed point iteration) with only 4 iterations. Closer runner up is the newton method with 5 iterations, followed by the secant method with 7 iterations and finally the bisection method. It isn't entirely fair to compare the methods like as they all have different starting requirements and they start at different positions.

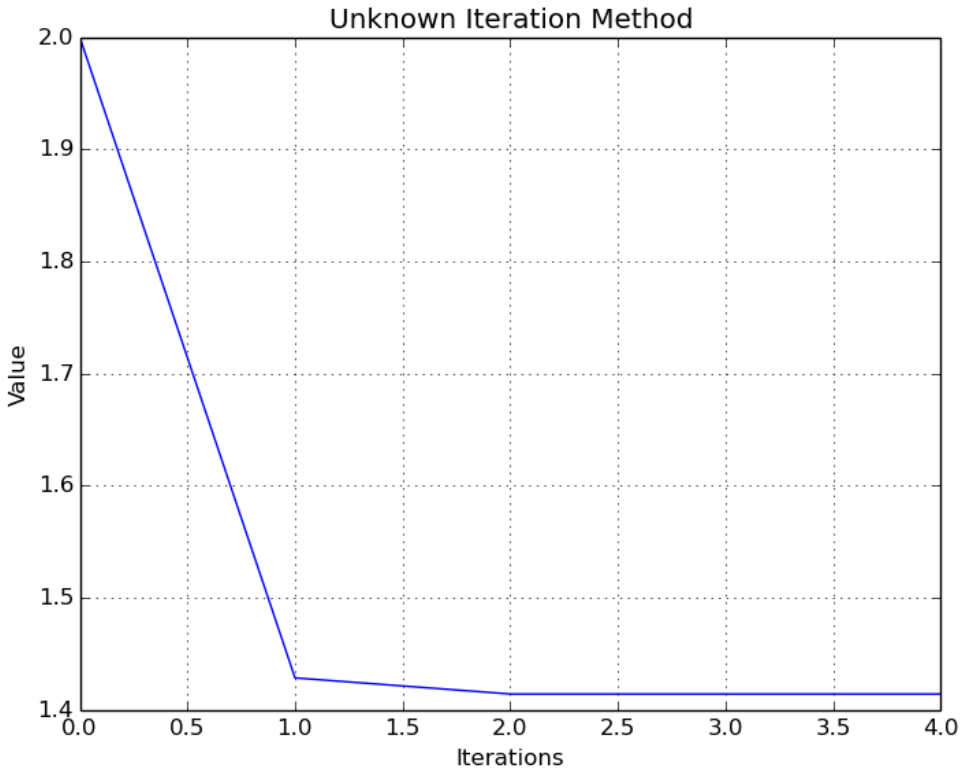
Figures

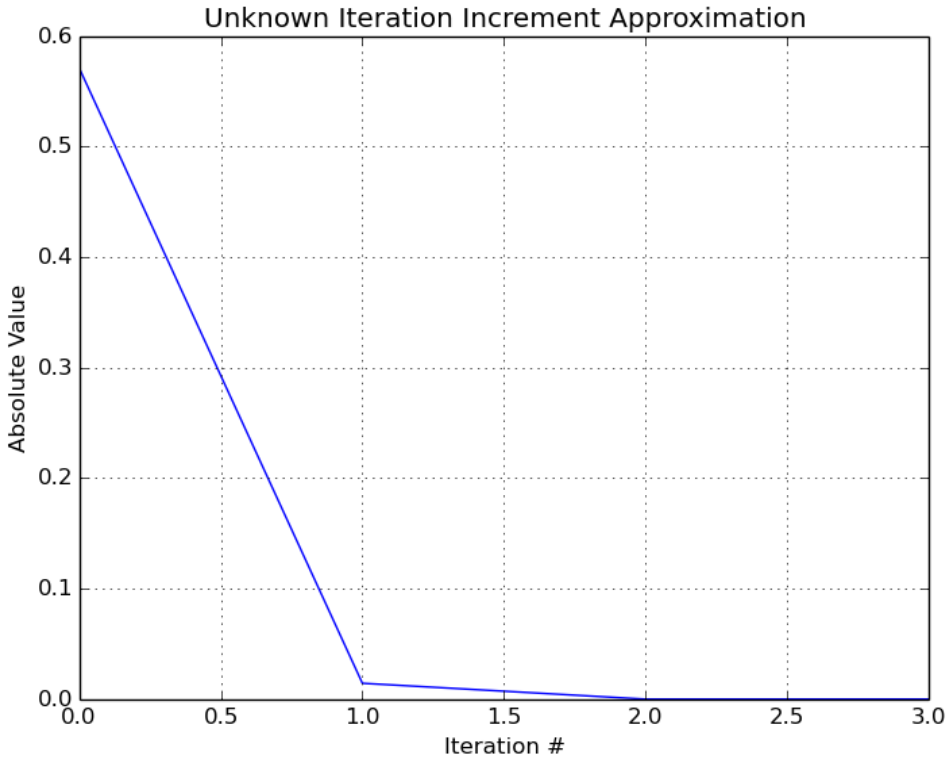


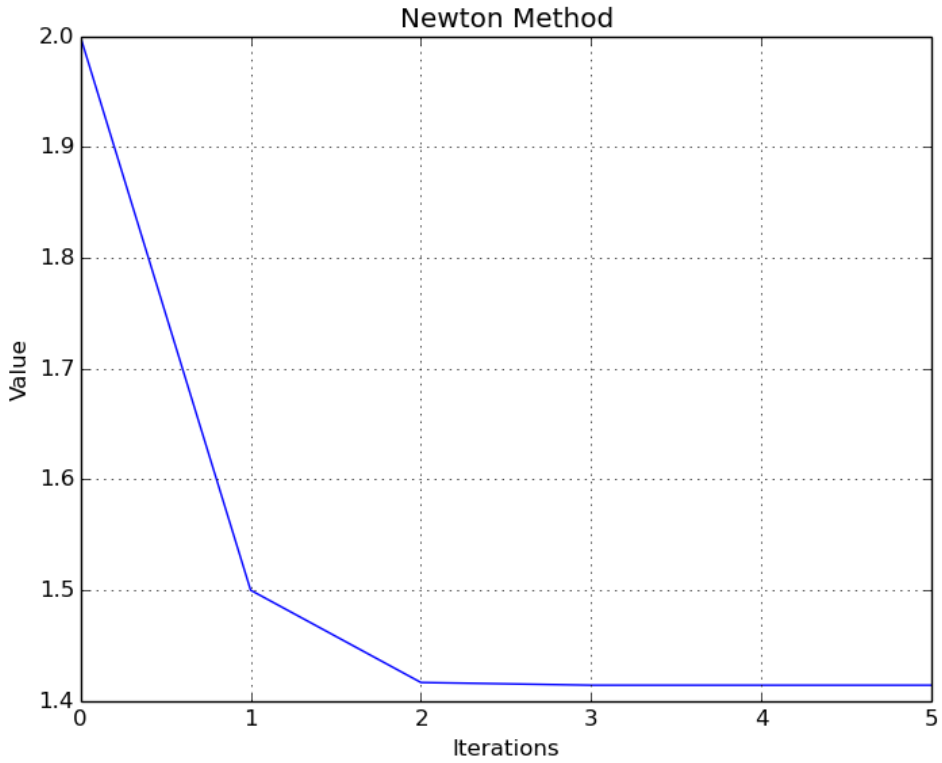
Function f(x) = x^2 -2

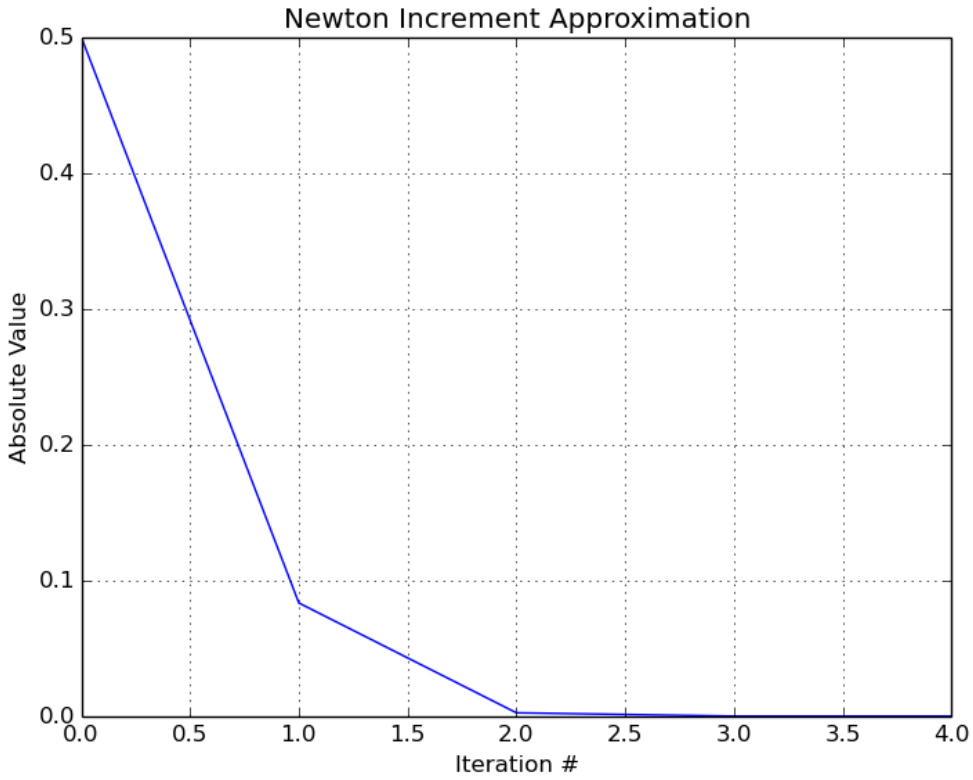

Bisec method
Value
Iterations
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
5
10
15
20
25
30

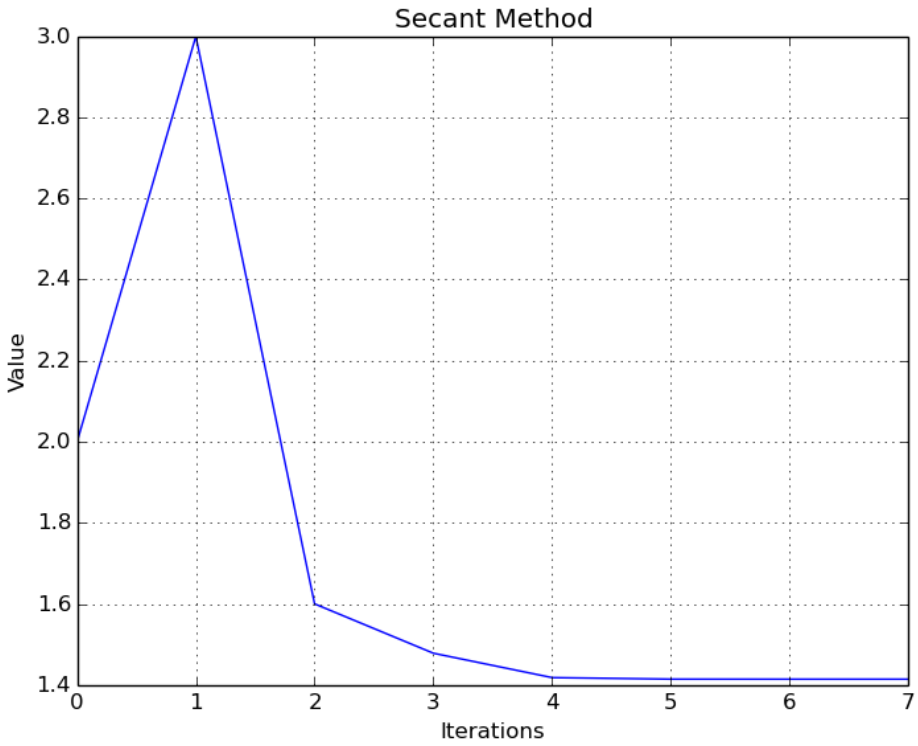

Bisec Increment Approximation
Absolute Value
Iteration #
0.5
0.4
0.3
0.2
0.1
0.0
0
5
10
15
20
25
30

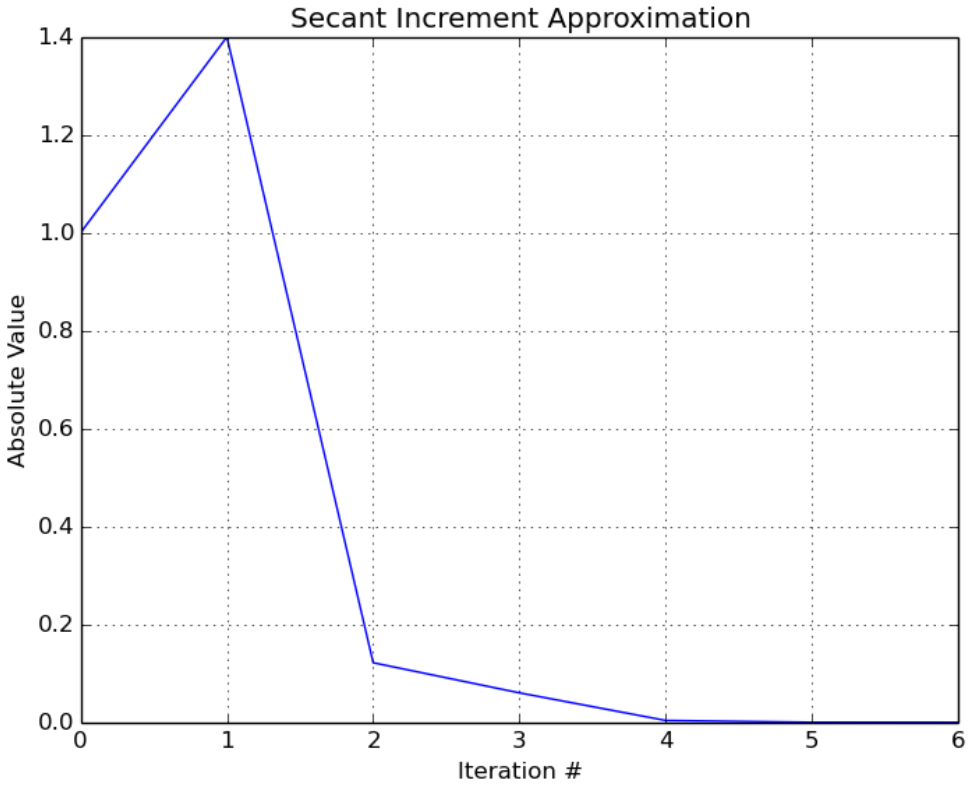

Unknown Iteration Method
Value
Iterations
2.0
1.9
1.8
1.7
1.6
1.5
1.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Unknown Iteration Increment Approximation
Absolute Value
Iteration #
0.6
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0


Newton Method
Value
Iterations
2.0
1.9
1.8
1.7
1.6
1.5
1.4
0
1
2
3
4
5


Newton Increment Approximation
Absolute Value
Iteration #
0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0


Secant Method
Value
Iterations
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
1.4
0
1
2
3
4
5
6
7


Secant Increment Approximation
Absolute Value
Iteration #
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 05 17:14:41 2015
@author: Hugo
"""
from pylab import *
from scipy import *
from scipy.misc import derivative
from scipy.optimize import fsolve


##----Assignment nr 1----##

## Task 1 a)##
def plotFunction(f):
     t = linspace(0, 2, 100)
     plot(t,f(t))
     grid(True)
     #title('Plot of function f(x) = x²-2')
     show()


## Task 1 b)
def bisec(f, a, b, tol):
     iterations = []
     i = 0
     a = float(a)
     b = float(b)
     if f(a)*f(b) > 0:
          raise Exception('This interval does not
contain the root or this function is not only
ascending or descending in this interval.')
     while True:
          i += 1
          c = (a + b) / 2
          sizeNewInterval = abs(a - c)
          val = f(c)
          iterations.append(c)
          if val*f(a) > 0:
```

```
                    a = c
          else:
               b = c
          if i > 1000:
                    raise Exception('Loop too
big... Something wrong!')
          if tol > sizeNewInterval or val == 0:
                    break
     return c, i, iterations


## Task 1 c)
def func(x):
     return x**2 -2


def funcThreeRoots(x):
     return (x - 0.2)**3 + (x - 0.2)**2 - x + 0.2


def funcWithNoRoot(x):
     return x**2 + 1


## Task 3 c)
def iterationMethod(f,x0,tol):
     i = 0
     x0 = float(x0)
     iterations = []
     iterations.append(x0)
     while True:
          i += 1
          x1 = (x0 * (x0**2 + 6) )/ (3 * x0**2 +
2)
          changePrevValue = abs(x1 - x0)
          x0 = x1
          iterations.append(x0)
          if tol > changePrevValue:
               if f(x0) < tol:
```

```
                    break
               if i > 100:
                         print 'With
iterationMethod a root was NOT found.'
                    return
     print 'With iterationMethod a root was found
at {} with {} iterations.'.format(x0, i)
     return x0, i, iterations
```

```
## Task 4
```

```
def newton(f,xn,tol):
     i = 0
     xn = float(xn)
     iterations = []
     iterations.append(xn)
     while True:
          i += 1
          val = f(xn)
          deriv = derivative(f,xn)
          if deriv == 0:
               raise Exception('A derivative
equal to zero???!')
          xnp1 = xn - val/deriv
          changePrevValue = abs(xn-xnp1)
          xn = xnp1
          iterations.append(xn)
          if tol > changePrevValue:
                    if f(xn) < tol:
                         print 'With newton
method a root was found at {} with {}
iterations.'.format(xn, i)
                    break
          if i > 100:
                    print 'Session timed out with
newton method, or root was NOT found.'
                    return
```

```
     return xn, i, iterations

def secant(f, x0, x1, tol):
     i = 0
     x0 = float(x0)
     x1 = float(x1)
     iterations = []
     iterations.extend([x0,x1])
     while True:
          i += 1
          val0 = f(x0)
          val1 = f(x1)
          x2 = x1 - (val1 * (x1 - x0))/(val1 -
val0)
          x0 = x1
          x1 = x2
          iterations.append(x1)
          interval = abs(x1-x0)
          if tol > interval or f(x1) < tol:
               print 'With secant method a root
was found at {} with {} iterations.'.format(x1,
i)
               break
          if i > 100:
               print 'Session timed out with
secant method, or root was NOT found.'
               return
     return x1, i, iterations


## bisec function without Exception when root not
found


def bisec2(f, a, b, tol):
        iterations = []
        i = 0
        a = float(a)
        b = float(b)
```

```
        if f(a)*f(b) > 0:
                raise Exception('This interval
does not contain the root or this function is not
only ascending or descending in this interval.')
        while True:
                i += 1
                c = (a + b) / 2
                sizeNewInterval = abs(a - c)
                val = f(c)
                iterations.append(c)
                if val*f(a) > 0:
                        a = c
                else:
                        b = c
                if i > 1000:
                        print 'Session timed out
with the bisect method, or a root was NOT found.'
                        return
                if tol > sizeNewInterval or val
== 0:
                        print 'With bisec method
a root was found at {} with {}
iterations'.format(c,i)
                break
        return c, i, iterations
```

```
#Task 5
```

```
def iterationPlot(name, iterations):
     plot(iterations)
     grid(True)
     title(name)
     xlabel('Iterations')
     ylabel('Value')
     show()
     return
```

```
def plotIncrementApproximation(name, iterations):
     tol = 1.e-14
     values = []
     i = 1
     val = 1
     while True:
          if i >= len(iterations) or val < tol:
               break
          val = abs(iterations[i] - iterations[i
- 1])
          values.append(val)
          i += 1
     plot(values)
     grid(True)
     title(name)
     xlabel('Iteration #')
     ylabel('Absolute Value')
     show()
     return
```

```
#Main program/code
tolerance = 1.e-8
```

```
mainmenu = str(input('For task 1-4 please press
1, for selection of specific iteration with
details (task 5) press 2: '))
while True:
     if mainmenu == str(1):
```

```
          plotFunction(func)
          print 'For function func, x**2 -2:'
          bisecReturn = bisec(func, 0, 2,
tolerance)
```

```
          print '--bisec method gives  {} in {}
iterations.'.format(bisecReturn[0],
bisecReturn[1])
          print '--fsolve method gives
{}.'.format(fsolve(func, 1.5)[0])
          print 'For function funcThreeRoots, (x
- 0.2)**3 + (x - 0.2)**2 - x + 0.2:'
          bisecReturn = bisec(funcThreeRoots, -1,
0.5, tolerance)
          print '--bisec method gives  {} in {}
iterations.'.format(bisecReturn[0],
bisecReturn[1])
          print '--fsolve method gives
{}.'.format(fsolve(funcThreeRoots, 0.3)[0])


          iterationMethod(func, 2, tolerance)
          iterationMethod(funcThreeRoots, 2,
tolerance)
          iterationMethod(funcWithNoRoot, 2,
tolerance)


          newton(func, 2, tolerance)
          newton(funcThreeRoots, 2, tolerance)
          newton(funcWithNoRoot, 2, tolerance)

          secant(func, 2, 3, tolerance)
          secant(funcThreeRoots, 2, 3, tolerance)
          secant(funcWithNoRoot, 2, 3, tolerance)
          break
     elif mainmenu == str(2):
          selection = str(input('For bisection
press 1, for unknown iteration method press 2,
for newton press 3, for secant press 4: '))
          while True:
               if selection == str(1):
                    bisecReturn = bisec2(func, 0,
2, tolerance)
```

```
                    iterationPlot('Bisec method',
bisecReturn[2])

plotIncrementApproximation('Bisec Increment
Approximation', bisecReturn[2])
                    break
               elif selection == str(2):
                    iterationReturn =
iterationMethod(func, 2, tolerance)

iterationPlot('Unknown Iteration Method',
iterationReturn[2])

plotIncrementApproximation('Unknown Iteration
Increment Approximation', iterationReturn[2])
                    break
               elif selection == str(3):
                              newtonReturn =
newton(func, 2, tolerance)

iterationPlot('Newton Method', newtonReturn[2])

plotIncrementApproximation('Newton Increment
Approximation', newtonReturn[2])
                    break
               elif selection == str(4):
                              secantReturn =
secant(func, 2, 3, tolerance)

iterationPlot('Secant Method', secantReturn[2])

plotIncrementApproximation('Secant Increment
Approximation', secantReturn[2])
                    break
               else:
                    selection = input('Please
type a value between 1 and 4: ')
          break
     else:
```

```
            mainmenu = str(input('Please type 1 or
2: '))
print 'Iterations complete.'
```