import numpy as np

from matplotlib import pyplot as plt

#define the tolerance to determine sufficient convergence

tol = 1.0e-12

#A boolean that is flagged false if items not desired to be printed

printWork = True

#Define the functions to be used to test the routines on

def f(x):

#return x\*\*2\*(1.-x)

return np.sin(x\*\*2) + 1.02 - np.exp(-x)

#return np.exp(x)- x - 1.000000005 Sueli & Mayers #1.5

#return np.exp(x) - x - 1 Sueli & Mayers #1.6

#Define the functions' derivatives as well

def fPrime(x):

#return 2\*x-3\*x\*\*2

return 2\*x\*np.cos(x\*\*2) + np.exp(-x)

#return np.exp(x) - 1 Sueli & Mayers #1.5

#return np.exp(x) - 1 Sueli & Mayers #1.6

########## ~~Definitions of the methods~~ ##########

def Newton(x\_0):

x = x\_0

x\_list = [x] #Create a list of the iterate values to make plotting easy

res\_list = [abs(f(x))] #create a residual list for plotting

res = abs(f(x)) #Note that the residues will be compared to the

k\_list = [0] #tolerance to determine convergence

n = 0 #initialize the iterator

while res > tol:

x = x - f(x)/(fPrime(x)) #The definition of Newton's method

res = abs(f(x)) #Update the residue before appending to the list

x\_list.append(x)

res\_list.append(res)

if len(x\_list) == 200: #Give a cutoff number of iterates to determine

#sufficient convergence

print 'Iteration did not converge within 200 iterates'

n = n+1 #NOTE: the iterator for the current iteration is updated

#at the end of the current iteration

k\_list.append(n)

break

n = n+1

k\_list.append(n) #Update the number of iterates after each iteration

if printWork:

fig = plt.figure(1) #Plot the x values against the iterates

plt.plot(k\_list,x\_list,'r^')

plt.title('x values vs the iterates for Newton\'s method')

plt.xlabel('k')

plt.ylabel('x value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_1.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(2) #Plot the residues against the iterates

plt.plot(k\_list,res\_list,'ys')

plt.title('residuals vs the iterates for Newton\'s method')

plt.xlabel('k')

plt.ylabel('residual value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_2.png',bbox\_inches = 'tight')

#plt.show

fig = plt.figure(3) #Plot the log of the residues against the iterates

plt.plot(k\_list,np.log10(res\_list),'bo')

plt.title('log of the residuals vs the iterates for Newton\'s method')

plt.xlabel('k')

plt.ylabel('log of residual')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_3.png',bbox\_inches = 'tight')

#plt.show()

print n+1 , x , res #iterates start at 0, so print n + 1 for total

def Secant(x\_0,x\_1): #Two initial values needed

x = x\_0

x\_list = [x]

res\_list = [abs(f(x))]

x = x\_1

x\_list.append(x)

res\_list.append(abs(f(x)))

res = abs(f(x))

k\_list = [0]

k\_list.append(1)

n = 1

while res > tol :

x = x - ((x-x\_list[n-1])/(f(x)-f(x\_list[n-1])))\*f(x)

res = abs(f(x))

x\_list.append(x)

res\_list.append(res)

if len(x\_list) == 200:

print 'Iteration did not converge within 200 iterates'

n = n+1

k\_list.append(n)

break

n = n+1

k\_list.append(n)

if printWork:

fig = plt.figure(4) #Plot the x values against the iterates

plt.plot(k\_list,x\_list,'r^')

plt.title('x values vs the iterates for Secant method')

plt.xlabel('k')

plt.ylabel('x value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_4.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(4) #Plot the residues against the iterates

plt.plot(k\_list,res\_list,'ys')

plt.title('residuals vs the iterates for Secant method')

plt.xlabel('k')

plt.ylabel('residual value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_5.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(6) #Plot the log of the residues against the iterates

plt.plot(k\_list,np.log10(res\_list),'bo')

plt.title('log of the residuals vs the iterates for Secant method')

plt.xlabel('k')

plt.ylabel('log of residual')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_6.png',bbox\_inches = 'tight')

#plt.show()

print n+1 , x , res #iterates start at 0, so print n+1

#to determine total iterations

def Bisection(a,b): #Like the secant method, need two initial values

c = (a + b)/2.0 #New value is average of initial two

c\_list = [c]

res\_list = [abs(f(c))]

res = abs(f(c))

k\_list = [0]

n = 0

while res > tol :

if f(a)\*f(c) < 0: #Condition if f(a) and f(c) are of different sign

b = c

c = (a + b)/2.0

c\_list.append(c)

res = abs(f(c))

res\_list.append(res)

else : #Condition if f(a) and f(c) are of same sign

a = c

c = (a + b)/2.0

c\_list.append(c)

res = abs(f(c))

res\_list.append(res)

if len(c\_list) == 200:

print 'Iteration did not converge within 200 iterates'

n += 1

k\_list.append(n)

break

n += 1

k\_list.append(n)

if printWork:

fig = plt.figure(7)

plt.plot(k\_list,c\_list,'r^')

plt.title('x values vs the iterates for Bisection method')

plt.xlabel('k')

plt.ylabel('x value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_7.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(8) #Plot the residues against the iterates

plt.plot(k\_list,res\_list,'ys')

plt.title('residuals vs the iterates for Bisection method')

plt.xlabel('k')

plt.ylabel('residual value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_8.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(9) #Plot the log of the residues against the iterates

plt.plot(k\_list,np.log10(res\_list),'bo')

plt.title('log of the residuals vs the iterates for Bisection method')

plt.xlabel('k')

plt.ylabel('log of residual')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_9.png',bbox\_inches = 'tight')

#plt.show()

print n , c , res

def Illinois(a,b): #Again, this method requires two initial values

c = (f(a)\*b - f(b)\*a)/(f(a)-f(b)) #Alternative definition from:

#https://en.wikipedia.org/wiki/False\_position\_method#The\_Illinois\_algorithm

c\_list = [c]

res = abs(f(c))

res\_list = [res]

n = 0

k\_list = [n]

side = 0 #flag that will attain values +/-1 depending on prev iteration

f\_b = f(b) #Cannot assign values to function calls, make vars instead

f\_a = f(a)

while res > tol:

if f(c)\*f(b) > 0:#So that next iteration encapsulates the fixed point

b = c

f\_b = f(c)#Change the function values accordingly

if side == -1:#flag raised if prev iteration f(a) was halved

f\_a = f\_a/2

c = (f\_a\*b - f\_b\*a)/(f\_a-f\_b)

c\_list.append(c)

res = abs(f(c))

if abs(res - tol) <= tol: #Bug fix for achieving a res of zero.

res = 1.0e-13 #res < tol anyway, so let res not == zero

res\_list.append(res)

side = -1

elif f(a)\*f(c) > 0:

a = c

f\_a = f(c)

if side == +1:

f\_b = f\_b/2

c = (f\_a\*b - f\_b\*a)/(f\_a-f\_b)

c\_list.append(c)

res = abs(f(c))

if abs(res - tol) <= tol:

res = 1.0e-13

res\_list.append(res)

side = +1

else:

print 'Iteration unexpectedly stopped. Check initial values'

break

if len(c\_list) == 200:

print 'Iteration did not converge within 200 iterates'

n += 1

k\_list.append(n)

break

n += 1

k\_list.append(n)

if printWork:

fig = plt.figure(10)

plt.plot(k\_list,c\_list,'r^') #plot the sequence values vs iterates

plt.title('x values vs the iterates for Illinois method')

plt.xlabel('k')

plt.ylabel('x value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_10.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(11) #Plot the residues against the iterates

plt.plot(k\_list,res\_list,'ys')

plt.title('residuals vs the iterates for Illinois method')

plt.xlabel('k')

plt.ylabel('residual value')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_11.png',bbox\_inches = 'tight')

#plt.show()

fig = plt.figure(12) #Plot the log of the residues against the iterates

plt.plot(k\_list,np.log10(res\_list),'bo')

plt.title('log of the residuals vs the iterates for Illinois method')

plt.xlabel('k')

plt.ylabel('log of residual')

plt.savefig('C:/Users/Parma\_Shon/Desktop/Math 514/HW/HW2/plot\_12.png',bbox\_inches = 'tight')

#plt.show()

print n+1 , c , res #updating iterates at end of iteration => n+1 total

########## ~~Run the routines/plot the functions~~ ##########

Newton(2.09)

Secant(2.0,2.09)

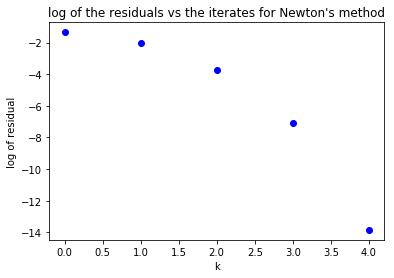
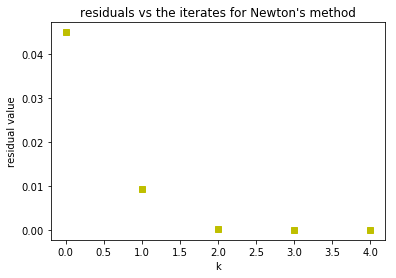
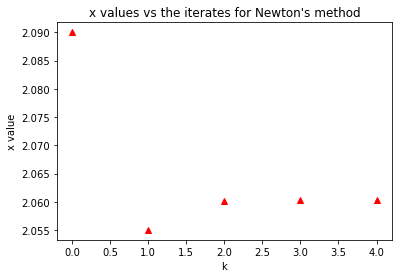
Bisection(2.0,2.09)

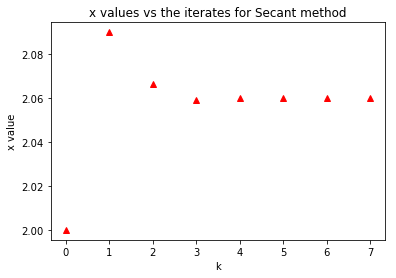
Illinois(2.0,2.09)

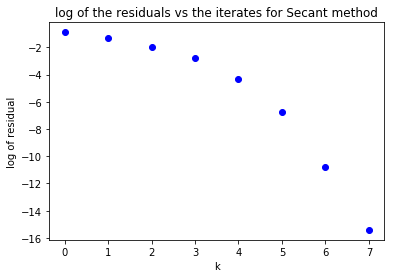
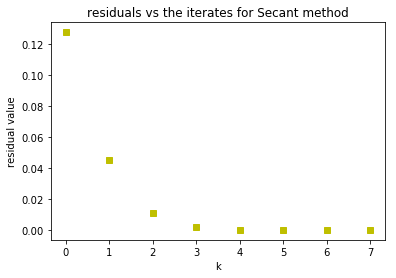
|  |  |  |  |
| --- | --- | --- | --- |
| Method | Number of iterates | Convergent value | Residue (1e-13 if formerly 0.0) |
| Newton | 5 | 2.0602482004 | 1.42663658664e-14 |
| Secant | 8 | 2.0602482004 | 3.88578058619e-16 |
| Bisection | 35 | 2.0602482004 | 5.83755266348e-13 |
| Illinois | 7 | 2.0602482004 | 1e-13 |

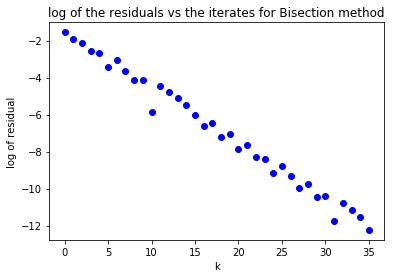
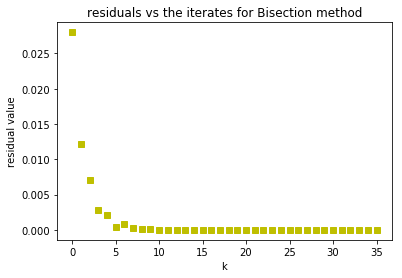
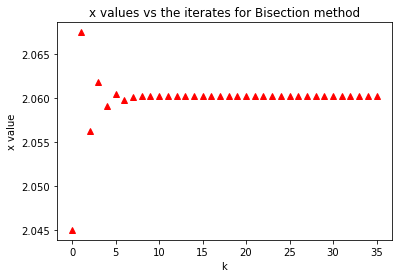
Table 1

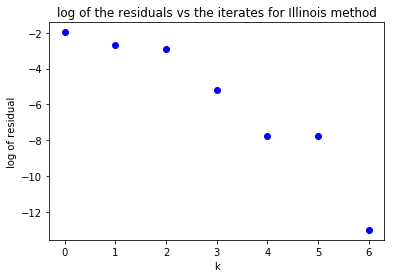
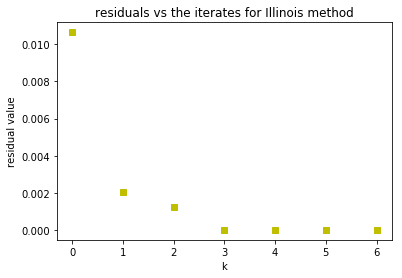
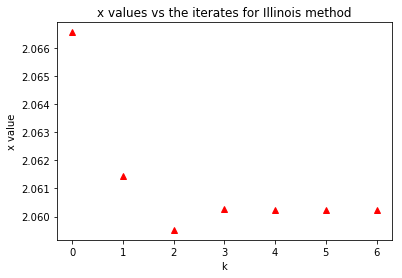
Plots











Discussion

First, we note the starting values of the 4 methods investigated. As can be seen in the final four lines of code, each code begins within a small neighborhood of ξ (approximately three-hundredths away). Thus, we would expect *every* method to converge quickly, which they do. The odd one out, however, is the bisection method; it takes 35 iterations to achieve ξ precise to 12 digits while the others are in the single digits. The logs of the residuals confirm this, as the bisection method is the only method for which a nearly straight line emerges from the data, rather than a ‘curve’ of negative curvature, which for these plots of log of residuals vs iterates indicates superlinear convergence for the other three methods, as we should suspect. The superlinear-ness of the other three methods does not appear to be equivalent; evidently the plot of the residuals vs the iterates for the Illinois method appears more linear than that of the corresponding plots of the Newton and Secant methods. Of course, comparing the number of iterates necessary to achieve a result accurate to twelve digits provides a different trend (besides the bisection method obviously.) From this standpoint, Newton’s method converges the fastest, taking only 5 iterations for the residuals to be under the tolerance. The Secant and Illinois methods are comparable in this context, with the Illinois method just a single iteration faster. Further analysis should be undertaken to estimate these asymptotic rates of convergence more accurately e.g. to verify quadratic convergence of Newton’s method (given how close the starting value is to ξ) to obtain a more complete picture of the relative rates of convergence. One could also experiment with starting points in much wider intervals away from ξ to determine a maximum neighborhood in which each of these methods will converge for a given function.