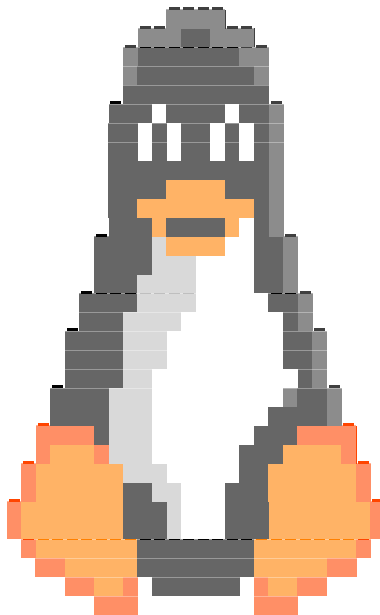

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΕΦΑΡΜΟΣΜΕΝΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ

ΦΥΣ 140 Εισαγωγή στην Επιστημονική Χρήση Υπολογιστών
Χειμερινό Εξάμηνο 2023

Φώτης Πτωχός και Αλέξανδρος Αττίκης
Φροντιστήριο 9

7 Νοεμβρίου 2023
15:00 - 17:00



Φροντιστήριο 9

Παράδειγμα 1 Εύρεση λύσης της μη γραμμικής εξίσωσης $f(x) = x^3 - 2x^2 + 2$ με τη μέθοδο Newton-Raphson, βήμα προς βήμα με γραφικές παραστάσεις:

tutorial9/ex1.py

```
1  #!/usr/bin/python3
2  '''
3  USAGE:
4      chmod +x ex3.py
5      python3 ex3.py
6      script -q ex3.log python3 -i ex3.py
7  '''
8  import sympy as s
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 def getNewtonRaphsonRoot(f, dfdx, x, epsilon, nStepsMax=100, debug=True):
13     '''
14     x_{n+1} = x_{n} - [ f(x_{n}) / f'(x_{n}) ]
15     '''
16     if debug:
17         print("=== ex1.py: getNewotonRaphsonRoot()")
18     nSteps = 0
19     roots = [x]
20     fVal = f(x)
21
22     while abs(f(x)) > epsilon and nSteps < nStepsMax:
23         try:
24             x = x - f(x) / dfdx(x)
25         except:
26             print("\tError! - derivative zero for x = ", x)
27             exit(1)
28         roots.append(x)
29         fVal = f(x)
30         nSteps+=1
31         if debug:
32             print("\tStep %d, x=%.6f, epsilon=%e" % (nSteps, roots[-1], fVal) )
33
34     # Either a solution is found, or too many iterations
35     if abs(fVal) > epsilon:
36         nSteps = -1
37     return roots, nSteps
38
39 def plotRoot(x, y, c="k", index=0):
40
41     plt.plot([x], [0], c+"o")
42     plt.plot([x, x], [0, y], c+":")
43     plt.text(x-0.12, -0.25, r"$\left(x_{%d}, 0\right)$" % (index), color=c)
44
45     plt.plot([x], [y], c+"o", fillstyle='none')
46     plt.text(x-0.12, y+0.25, r"$\left(x_{%d}, f(x_{%d})\right)$" % (index,
47 index), color=c)
```

```

47     return
48
49
50 # Define symbolic variable and expression
51 x = s.Symbol("x")
52 F = x**3 - 2*x**2 + 2
53 DF = s.diff(F, x)
54
55 # Create Python function f and df from SymPy expressions F and DF (evaluated
    numerically)
56 f = s.lambdify(x, F, 'numpy')
57 df = s.lambdify(x, DF, 'numpy')
58 eq = s.Eq(F, 0)
59 exactRoots = s.solve(eq, x)
60 realRoots = [root.evalf() for root in exactRoots if root.is_real]
61
62 print("=== Function and derivative:\n\tf(x) = ", F)
63 print("\tdf(x) = ", DF)
64
65 # Define various variables
66 xSeed = 1.5
67 epsilon = 1e-06
68 nSteps = 3 # number of iterations
69 rootList = [xSeed] # first root is the seed
70 xList = np.arange(-1.0, 3.5, 0.0001) # np.linspace(-0.5, 2.3, 400)
71 fxList = f(xList)
72 cList = ["k", "g", "m", "c", "y"]
73 saveName = "ex1_x0%s" % (str(xSeed).replace(".", "p"))
74 np_Roots, np_Steps = getNewtonRaphsonRoot(f, df, xSeed, epsilon)
75 np_X = np_Roots[-1]
76 if np_Steps > 0:
77     print("=== ex1.py\n\tFound root to be '%.3f' after %d iterations!" % (np_X
        , np_Steps))
78 else:
79     print("=== ex1.py\n\tSolution not found!")
80
81
82 print("=== ex1.py: Plot")
83 # Create a figure with subplots
84 plt.figure(figsize=(16, 6)).suptitle("Newton-Raphson Method")
85 #plt.figure(figsize=(16, 6)).suptitle(r"$f(x) = %s$, $f'(x) = %s$" % (s.
    latex(F), s.latex(DF)))
86
87 # Perform Newton-Raphson iterations
88 for i in range(nSteps):
89
90     # Coordinates of the root
91     x0 = rootList[-1]
92
93     # Construct the tangent equation  $y_{\text{tan}} = f(x_0) + f'(x_0)(x - x_0)$ 
94     tList = [f(x0) + df(x0) * (x - x0) for x in xList]
95
96     # Find the next root using Newton-Raphson

```

```

97     xi = x0 - f(x0) / df(x0)
98     rootList.append(xi)
99     print("\tStep %d, x_{%d}=%.3f, x_{%d} = %.3f - f(%.3f) / df(%.3f) = %.3f" %
100           (i+1, i, x0, i+1, x0, x0, xi) )
101
102     # Create each plot on a separate pad
103     plt.subplot(1, nSteps, i+1)
104
105     # Plot the function we are trying to find the root of
106     plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=s$' % s.latex(F) )
107
108     # Plot the current root
109     plotRoot(x0, f(x0), cList[i], i)
110
111     # Plot the tangent point/line
112     plt.plot(xList, tList, "r--", label=r"slope $f\,'(x_{%d})$" % (i) )
113
114     # Add y=0
115     plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")
116
117     # Plot the next root estimate
118     plotRoot(xi, f(xi), cList[i+1], i+1)
119
120     # Customise axes
121     plt.title(f'Step {i+1}')
122     plt.xlabel('x')
123     plt.ylabel('f(x)')
124     plt.ylim(-1, 6)
125     plt.legend(loc='upper left')
126     plt.grid(False)
127     plt.tight_layout()
128     for ext in [".png", ".pdf"]:
129         plt.savefig(saveName + ext)
130
131     # Create a figure with the roots
132     plt.figure(figsize=(16, 6))
133     plt.plot(np_Roots, "ro-", lw=2, label=r'root $x_{n}$')
134     plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")
135     for i in range(nSteps+1):
136         plt.plot([i], np_Roots[i], cList[i]+"o", label=r'$x_{%d}$' % (i))
137     plt.xlabel('step')
138     plt.ylabel(r'$x_{n}$')
139     plt.ylim(-1.5, +2.5)
140     plt.title('Newton-Raphson Method')
141     plt.grid(True)
142     plt.legend(loc='upper right', title=r'$N=%d, \epsilon=0.1e$' % (np_Steps,
143         epsilon) )
144     for ext in [".png", ".pdf"]:
145         plt.savefig(saveName + "-roots" + ext)
146     plt.show()

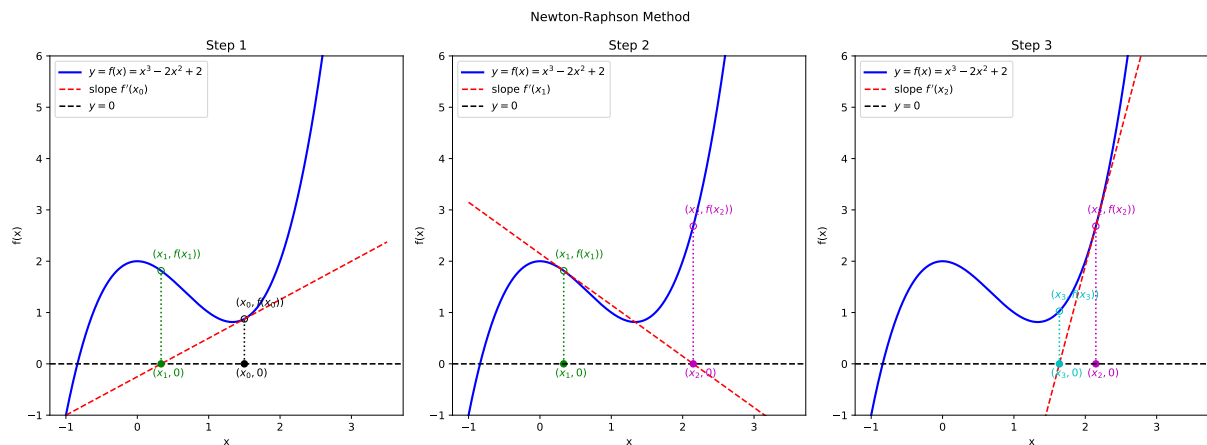
```

Αποτέλεσμα:

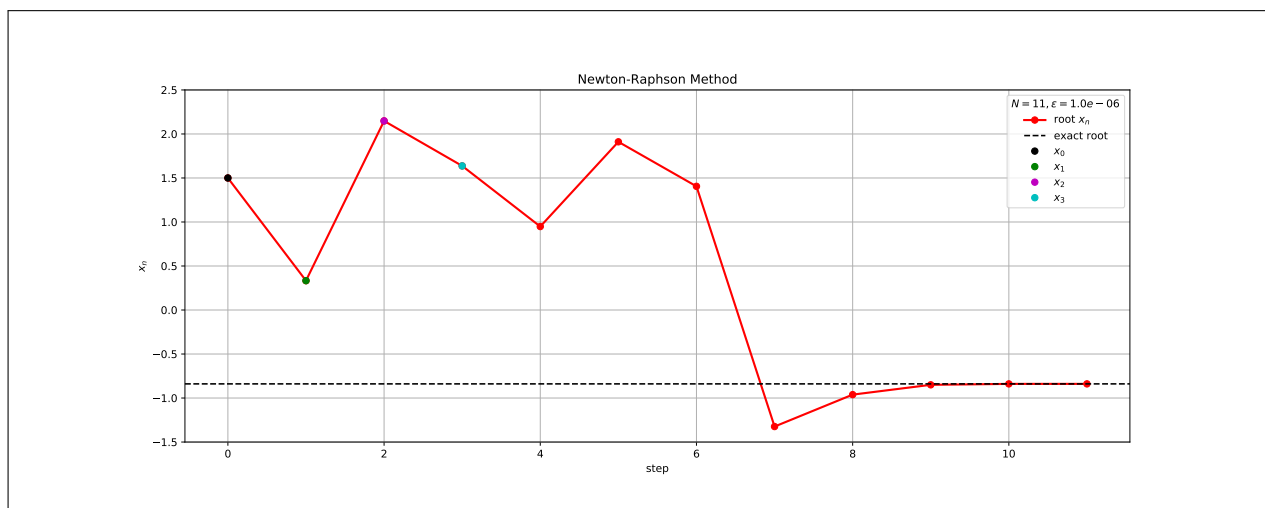
```

=== Function and derivative:
  f(x) = x**3 - 2*x**2 + 2
  df(x) = 3*x**2 - 4*x
=== ex1.py: getNewotonRaphsonRoot()
  Step 1, x=0.333333, epsilon=1.814815e+00
  Step 2, x=2.148148, epsilon=2.683636e+00
  Step 3, x=1.637080, epsilon=1.027363e+00
  Step 4, x=0.948393, epsilon=1.054133e+00
  Step 5, x=1.910874, epsilon=1.674562e+00
  Step 6, x=1.405090, epsilon=8.254823e-01
  Step 7, x=-1.324018, epsilon=-3.827083e+00
  Step 8, x=-0.961438, epsilon=-7.374450e-01
  Step 9, x=-0.850022, epsilon=-5.924849e-02
  Step 10, x=-0.839381, epsilon=-5.140478e-04
  Step 11, x=-0.839287, epsilon=-3.988341e-08
=== ex1.py
  Found root to be '-0.839' after 11 iterations!
=== ex1.py: Plot
  Step 1, x_{0}=1.500, x_{1} = 1.500 - f(1.500) / df(1.500) = 0.333
  Step 2, x_{1}=0.333, x_{2} = 0.333 - f(0.333) / df(0.333) = 2.148
  Step 3, x_{2}=2.148, x_{3} = 2.148 - f(2.148) / df(2.148) = 1.637

```



Παράδειγμα 1 συνεχίζεται. . .



Παράδειγμα 2 Εύρεση λύσης της μη γραμμικής εξίσωσης $f(x) = x^3 - 2x^2 + 2$ με τη μέθοδο Secant, βήμα προς βήμα με γραφικές παραστάσεις:

tutorial9/ex2.py

```
1  #!/usr/bin/python3
2  '''
3  USAGE:
4      chmod +x ex3.py
5      python3 ex3.py
6      script -q ex3.log python3 -i ex3.py
7  '''
8  import sympy as s
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 def secant(f, x0, x1):
13     x = None
14     try:
15         num = f(x1) * (x1 - x0)
16         den = f(x1) - f(x0)
17         x = x1 - num/den
18     except:
19         print("\tError! Denominator is zero for x0 = %s, x1 = %s" % (x0, x1) )
20         exit(1)
21     return x
22
23 def getSecantRoot(f, x0, x1, epsilon, nStepsMax=100, debug=True):
24     '''
25     x_{n+1} = x_{n} - f(x_{n}) [ (x_{n} - x_{n-1}) / ( f(x_{n}) - f(x_{n-1}) ) ]
26     '''
27     if debug:
28         print("=== ex2.py: getNewotonRaphsonRoot()")
29     nSteps = 0
30     roots = [x0, x1]
31     fVal0 = f(x0)
32     fVal1 = f(x1)
33
34     while abs(fVal1) > epsilon and nSteps < nStepsMax:
35         try:
36             num = fVal1 * (x1 - x0)
37             den = fVal1 - fVal0
38             x = x1 - num/den
39         except:
40             print("\tError! Denominator is zero for x0 = %s, x1 = %s" % (x0, x1) )
41             exit(1)
42             roots.append(x)
43
44         if debug:
45             print("\tStep %d, x0=%.6f, x1=%.6f, x2=%.6f, epsilon=%e" % (nSteps,
46                 x0, x1, x, fVal1) )
46             # Re-set values
```

```

47     x0 = x1
48     x1 = x
49     fVal0 = fVal1
50     fVal1 = f(x1)
51     nSteps+=1
52
53     # Either a solution is found, or too many iterations
54     if abs(fVal1) > epsilon:
55         nSteps = -1
56     return roots, nSteps
57
58 def plotRoot(x, y, c="k", index=0):
59
60     plt.plot([x], [0], c+"o")
61     plt.plot([x, x], [0, y], c+":")
62     plt.text(x-0.12, -0.25, r"$\left(x_{%d}, 0\right)$" % (index), color=c)
63
64     plt.plot([x], [y], c+"o", fillstyle='none')
65     plt.text(x-0.12, y+0.25, r"$\left(x_{%d}, f(x_{%d})\right)$" % (index,
66 index), color=c)
67     return
68
69 # Define symbolic variable and expression
70 x = s.Symbol("x")
71 F = x**3 - 2*x**2 + 2
72
73 # Create Python function f and df from SymPy expression F
74 f = s.lambdify(x, F, 'numpy')
75 eq = s.Eq(F, 0)
76 exactRoots = s.solve(eq, x)
77 realRoots = [root.evalf() for root in exactRoots if root.is_real]
78
79 # Define various variables
80 x0 = 2.0 # 2.2 # 2.7 #2.0
81 x1 = 1.3 # 1.8 # x0-1.0 #x0-0.8
82 epsilon = 1e-06
83 nSteps = 3
84 rootList = [x0, x1] # new two seeds
85 xList = np.arange(-1.0, 3.5, 0.0001)
86 fxList = f(xList)
87 cList = ["k", "g", "m", "c", "y"]
88 saveName = "ex2_x0%s_x1%s" % (str(x0).replace(".", "p"), str(x1).replace(".",
89 "p"))
90 sec_Roots, sec_Steps = getSecantRoot(f, x0, x1, epsilon)
91 sec_X = sec_Roots[-1]
92 if sec_Steps > 0:
93     print("=== ex2.py\n\tFound root to be '%.3f' after %d iterations!" % (
94 sec_X, sec_Steps))
95 else:
96     print("=== ex2.py\n\tSolution not found!")

```



```

97 print("=== ex2.py: Plot")
98 # Create a figure with subplots
99 plt.figure(figsize=(16, 6)).suptitle("Secant Method")
100
101 # Perform iterations
102 for i in range(0, nSteps, 1):
103
104     # Coordinates of the root
105     x0 = rootList[i]
106     x1 = rootList[i+1]
107     x2 = secant(f, x0, x1) # find the next root using Secant method
108     rootList.append(x2)
109
110     # Construct the tangent equation  $y_{\text{tan}} = f(x_0) + f'(x_0)(x-x_0)$ 
111     tList = [f(x0) + (f(x1) - f(x0)) * (x - x0) / (x1 - x0) for x in xList]
112     print("\tStep %d,  $x_{\text{%d}} = %.2f$ ,  $x_{\text{%d}} = %.2f$ ,  $x_{\text{%d}} = %.2f$ " % (i+1, i, x0,
113         i+1, x1, i+2, x2) )
114
115     # Create each plot on a separate pad
116     plt.subplot(1, nSteps, i+1)
117
118     # Plot the function we are trying to find the root of
119     plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=s$' % (s.latex(F)) )
120
121     # Plot the current root
122     plotRoot(x0, f(x0), cList[i], i)
123     plotRoot(x1, f(x1), cList[i], i+1)
124
125     # Plot the tangent point/line
126     plt.plot(xList, tList, "r--", label=r"chord  $x_{\text{%d}}$  and  $x_{\text{%d}}$ " % (i, i
127         +1) )
128
129     # Add y=0
130     plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")
131
132     # Plot the next root estimate
133     plotRoot(x2, f(x2), cList[i+1], i+2)
134
135     # Customise axes
136     plt.title(f'Step {i+1}')
137     plt.xlabel('x')
138     plt.ylabel('f(x)')
139     plt.ylim(-1, 4)
140     plt.legend(loc='upper left')
141     plt.grid(False)
142
143     plt.tight_layout()
144     for ext in [".png", ".pdf"]:
145         plt.savefig(saveName + ext)
146
147 # Create a figure with the roots
148 plt.figure(figsize=(16, 6))
149 plt.plot(sec_Roots, "ro-", lw=2, label=r'root  $x_{\text{%d}}$ ')
150 plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")

```

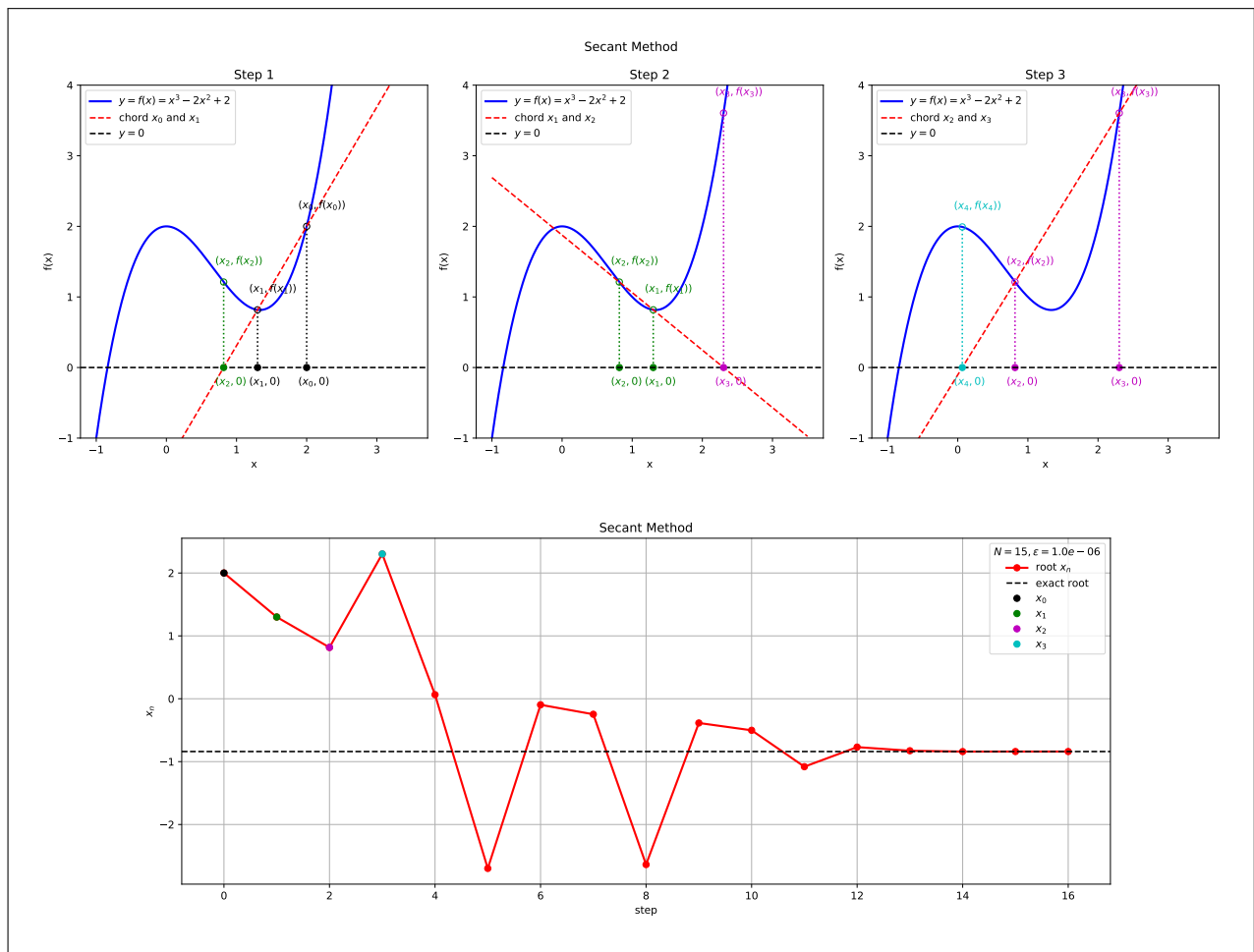
Παράδειγμα 2 συνεχίζεται...

```
148 for i in range(nSteps+1):
149     plt.plot([i], sec_Roots[i], cList[i]+"o", label=r'$x_{%d}$' % (i))
150 plt.xlabel('step')
151 plt.ylabel(r'$x_{n}$')
152 plt.title('Secant Method')
153 plt.grid(True)
154 plt.legend(loc='upper right', title=r'$N=%d, \epsilon=%.1e$' % (sec_Steps,
    epsilon) )
155 for ext in [".png", ".pdf"]:
156     plt.savefig(saveName + "-roots" + ext)
157 plt.show()
```

Αποτέλεσμα:

```
=== ex2.py: getNewtonRaphsonRoot()
Step 0, x0=2.000000, x1=1.300000, x2=0.816568, epsilon=8.170000e-01
Step 1, x0=1.300000, x1=0.816568, x2=2.302683, epsilon=1.210907e+00
Step 2, x0=0.816568, x1=2.302683, x2=0.064884, epsilon=3.604928e+00
Step 3, x0=2.302683, x1=0.064884, x2=-2.698390, epsilon=1.991853e+00
Step 4, x0=0.064884, x1=-2.698390, x2=-0.096042, epsilon=-3.221042e+01
Step 5, x0=-2.698390, x1=-0.096042, x2=-0.246794, epsilon=1.980666e+00
Step 6, x0=-0.096042, x1=-0.246794, x2=-2.636966, epsilon=1.863154e+00
Step 7, x0=-0.246794, x1=-2.636966, x2=-0.385496, epsilon=-3.024355e+01
Step 8, x0=-2.636966, x1=-0.385496, x2=-0.501673, epsilon=1.645498e+00
Step 9, x0=-0.385496, x1=-0.501673, x2=-1.080381, epsilon=1.370388e+00
Step 10, x0=-0.501673, x1=-1.080381, x2=-0.769066, epsilon=-1.595492e+00
Step 11, x0=-1.080381, x1=-0.769066, x2=-0.826664, epsilon=3.622016e-01
Step 12, x0=-0.769066, x1=-0.826664, x2=-0.840057, epsilon=6.833459e-02
Step 13, x0=-0.826664, x1=-0.840057, x2=-0.839279, epsilon=-4.217661e-03
Step 14, x0=-0.840057, x1=-0.839279, x2=-0.839287, epsilon=4.425810e-05
=== ex2.py
Found root to be '-0.839' after 15 iterations!
=== ex2.py: Plot
Step 1, x_{0}=2.00, x_{1} = 1.30, x_{2} = 0.82
Step 2, x_{1}=1.30, x_{2} = 0.82, x_{3} = 2.30
Step 3, x_{2}=0.82, x_{3} = 2.30, x_{4} = 0.06
```

Παράδειγμα 2 συνεχίζεται. . .



Παράδειγμα 3 Απλά παραδείγματα για βασικές πράξεις μεταξύ συμβόλων με τη βοήθεια της SymPy:

tutorial9/ex3.py

```
1  #!/usr/bin/python3
2  '''
3  USAGE:
4      chmod +x ex3.py
5      python3 ex3.py
6      script -q ex3.log python3 -i ex3.py
7  '''
8  import sympy as s
9
10 # Define symbolic variables
11 x, y = s.symbols('x y')
12
13
14 # Function definitions
15 f = s.sin(x)/x
16 print("Function f(x):\n\t", f)
17
18
19 # Differentiation / Integration wrt x
20 df_dx = s.diff(f, x)
21 intx_f = s.integrate(df_dx, x)
22 print("Differentiation df(x)/dx:\n\t", df_dx)
23 print("Integral of df_dx:\n\t", intx_f)
24
25
26 # Limit evaluation (in SymPy the symbol s.oo represent positive infinity)
27 lim_x_0 = s.limit(f, x, 0) # Limit of f as x approaches 2
28 lim_x_inf = s.limit(f, x, s.oo) # Limit of 1/f as x approaches infinity
29 print("Limit as x -> 0:\n\t", lim_x_0)
30 print("Limit as x -> infty:\n\t", lim_x_inf)
31 # L'Hopital rule:  $\lim_{x \rightarrow c} f(x) / g(x) == \lim_{x \rightarrow c} f'(x) / g'(x)$ 
32
33
34 # Equation solving
35 eq = s.Eq(f, 0) # Define the equation f = 0
36 solutions = s.solve(eq, x) # Solve for x in the equation
37 print("Solutions of f(x) = 0:\n\t", solutions)
38
39
40 # Taylor expansion around x=0, up to 5 terms
41 taylor_expansion_f = s.series(f, x, 0, 5)
42 print("Taylor expansion of f(x):\n\t", taylor_expansion_f)
```

Αποτέλεσμα:

```
Function f(x):  
    sin(x)/x  
Differentiation df(x)/dx:  
    cos(x)/x - sin(x)/x**2  
Integral of df_dx:  
    sin(x)/x  
Limit as x -> 0:  
    1  
Limit as x -> infty:  
    0  
Solutions of f(x) = 0:  
    [pi]  
Taylor expansion of f(x):  
    1 - x**2/6 + x**4/120 + O(x**5)
```

Παράδειγμα 4 Παράδειγμα εύρεσης ριζών με τη μέθοδο διχοτόμησης Bisection method:

tutorial9/ex4.py

```
1  #!/usr/bin/python3
2  '''
3  USAGE:
4      chmod +x ex3.py
5      python3 ex3.py
6      script -q ex3.log python3 -i ex3.py
7  '''
8  import sympy as s
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 def bisection(f, xL, xR):
13     fxL = f(xL)
14     fxR = f(xR)
15     x    = (xL + xR) / 2
16     fx   = f(x)
17
18     if (fx * fxL > 0):
19         xL = x
20         fxL = f(xL)
21     else:
22         xR = x
23
24     # Find the new midpoint
25     x    = (xL + xR) / 2
26     fx   = f(x)
27     return x, xL, xR
28
29
30 def getBisectionRoots(f, xL, xR, epsilon, nStepsMax=100, debug=True):
31     '''
32     Finds the root of the function f within the interval [xL, xR] using the
33     Bisection method.
34     '''
35     if debug:
36         print("=== ex4.py")
37
38     fxL = f(xL)
39     fxR = f(xR)
40
41     if fxL * fxR > 0:
42         print("Error! Function does not have opposite signs at interval
43         endpoints [!"]
44         exit(1)
45
46     nSteps = 1
47     roots  = []
48     x      = (xL + xR) / 2
49     fx     = f(x)
```

```

49     while abs(fx) > epsilon and nSteps < nStepsMax:
50
51         if (fx * fxL > 0):
52             # same sign
53             xL = x
54             fxL = f(xL)
55         else:
56             # opposite sign
57             xR = x
58
59         # Find the new midpoint
60         x = (xL + xR) / 2
61         fx = f(x)
62         roots.append(x)
63         if debug:
64             print("\tStep %d, xL=%.6f, xR=%.6f, epsilon=%e" % (nSteps, xL, xR,
65                 (xR - xL) / 2))
66             nSteps += 1
67
68     return roots, nSteps
69
70 def plotRoot(x, y, c, subscript):
71
72     plt.plot([x], [0], c+"o")
73     plt.plot([x, x], [0, y], c+":")
74     plt.text(x-0.12, +0.25, r"$\left(x_{%s}, 0\right)$" % (subscript), color=c)
75
76     plt.plot([x], [y], c+"o", fillstyle='none')
77     plt.text(x-0.12, y+0.25, r"$\left(x_{%s}, f(x_{%s})\right)$" % (subscript,
78         subscript), color=c)
79     return
80
81 # Define symbolic variable and expression
82 x = s.Symbol("x")
83 F = x**3- 2*x**2 + 2
84 f = s.lambdify(x, F, 'numpy')
85 eq = s.Eq(F, 0)
86 exactRoots = s.solve(eq, x)
87 realRoots = [root.evalf() for root in exactRoots if root.is_real]
88
89 # Define various variables
90 xL = -1.3
91 xR = +1.3
92 x = (xL + xR) / 2
93 epsilon = 1e-06
94 nSteps = 3
95 rootList = []
96 xList = np.arange(xL, xR, 0.0001)
97 fxList = f(xList)
98 cList = ["k", "g", "m", "c", "y"]
99 bi_Roots, bi_Steps = getBisectionRoots(f, xL, xR, epsilon)

```

```

100 # x, xL, xR = bisection(f, xL, xR)
101 if nSteps > 0:
102     print( "=== ex4.py\n\tFound root to be '%.3f' after %d iterations!" % (
        bi_Roots[-1], bi_Steps) )
103 else:
104     print("=== ex4.py\n\tSolution not found!")
105
106
107 print("=== ex4.py: Plot")
108 # Create a figure with subplots
109 plt.figure(figsize=(16, 6)).suptitle("Bisection Method")
110
111 # Perform Newton-Raphson iterations
112 for i in range(nSteps):
113
114     # Create each plot on a separate pad
115     plt.subplot(1, nSteps, i+1)
116
117     # Plot the function we are trying to find the root of
118     plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=%s$' % s.latex(F))
119
120     # Add y=0
121     plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")
122
123     # Plot the current and next roots
124     plotRoot(xL, f(xL), cList[i], "L")
125     plotRoot(xR, f(xR), cList[i], "R")
126     plotRoot(x, f(x), cList[i+1], "")
127     rootList.append(x)
128     print("\tStep %d/%d, xL=%.3f, xR = %.3f, x = %.3f" % (i+1, nSteps, xL, xR,
        x) )
129
130     # Find the next root using Newton-Raphson
131     x, xL, xR = bisection(f, xL, xR)
132
133     # Customise axes
134     plt.title(f'Step {i+1}')
135     plt.xlabel('x')
136     plt.ylabel('f(x)')
137     plt.ylim(-4.0, 2.5)
138     plt.legend(loc='lower right')
139     plt.grid(True)
140
141 plt.tight_layout()
142 for ext in [".png", ".pdf"]:
143     plt.savefig("ex4" + ext)
144
145 # Create a figure with the roots
146 plt.figure(figsize=(16, 6))
147 plt.plot(bi_Roots, "ro-", lw=2, label=r'root $x_{n}$')
148 plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")
149 for i in range(nSteps+1):
150     plt.plot([i], bi_Roots[i], cList[i]+"o", label=r'$x_{%d}$' % (i))

```


Παράδειγμα 4 συνεχίζεται...

```
151 plt.xlabel('step')
152 plt.ylabel(r'$x_{n}$')
153 plt.title("Bisection Method")
154 plt.grid(True)
155 plt.legend(loc='upper right', title=r'$N=%d$, \epsilon=%.1e$' % (bi_Steps,
    epsilon) )
156 for ext in [".png", ".pdf"]:
157     plt.savefig("ex4-roots" + ext)
158 plt.show()
```

Αποτέλεσμα:

```
=== ex4.py
Step 1, xL=1.750000, xR=3.000000, epsilon=6.250000e-01
Step 2, xL=2.375000, xR=3.000000, epsilon=3.125000e-01
Step 3, xL=2.375000, xR=2.687500, epsilon=1.562500e-01
Step 4, xL=2.531250, xR=2.687500, epsilon=7.812500e-02
Step 5, xL=2.531250, xR=2.609375, epsilon=3.906250e-02
Step 6, xL=2.570312, xR=2.609375, epsilon=1.953125e-02
Step 7, xL=2.589844, xR=2.609375, epsilon=9.765625e-03
Step 8, xL=2.589844, xR=2.599609, epsilon=4.882812e-03
Step 9, xL=2.589844, xR=2.594727, epsilon=2.441406e-03
Step 10, xL=2.592285, xR=2.594727, epsilon=1.220703e-03
Step 11, xL=2.593506, xR=2.594727, epsilon=6.103516e-04
Step 12, xL=2.594116, xR=2.594727, epsilon=3.051758e-04
Step 13, xL=2.594116, xR=2.594421, epsilon=1.525879e-04
Step 14, xL=2.594269, xR=2.594421, epsilon=7.629395e-05
Step 15, xL=2.594269, xR=2.594345, epsilon=3.814697e-05
Step 16, xL=2.594307, xR=2.594345, epsilon=1.907349e-05
Step 17, xL=2.594307, xR=2.594326, epsilon=9.536743e-06
Step 18, xL=2.594307, xR=2.594316, epsilon=4.768372e-06
Step 19, xL=2.594312, xR=2.594316, epsilon=2.384186e-06
Step 20, xL=2.594312, xR=2.594314, epsilon=1.192093e-06
Step 21, xL=2.594313, xR=2.594314, epsilon=5.960464e-07
Step 22, xL=2.594313, xR=2.594314, epsilon=2.980232e-07
Step 23, xL=2.594313, xR=2.594313, epsilon=1.490116e-07
=== ex4.py
Found root to be '2.594' after 24 iterations!
=== ex4.py: Plot
Step 1/3, xL=0.500, xR = 3.000, x = 1.750
Step 2/3, xL=1.750, xR = 3.000, x = 2.375
Step 3/3, xL=2.375, xR = 3.000, x = 2.688
```

