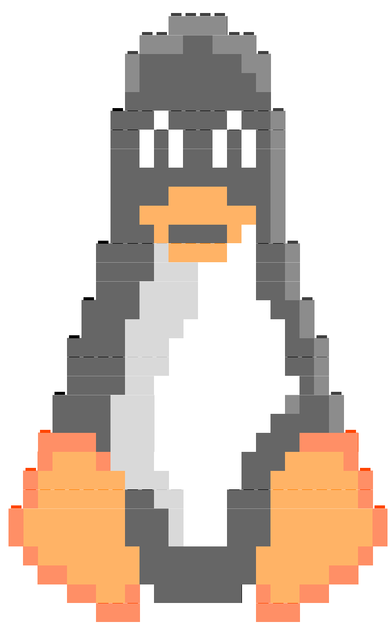
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΕΦΑΡΜΟΣΜΕΝΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ

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

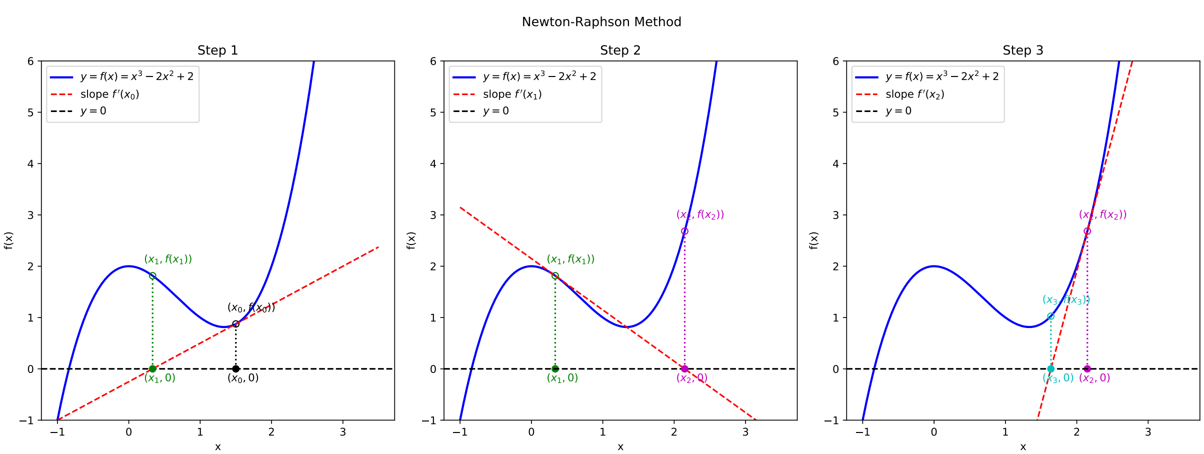
```
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex3.py
   python3 ex3.py
   script -q ex3.log python3 -i ex3.py
'''
import sympy as s
import numpy as np
import matplotlib.pyplot as plt

def getNewtonRaphsonRoot(f, dfdx, x, epsilon, nStepsMax=100, debug=True):
    '''
    x_{n+1}  = x_{n} - [ f(x_{n}) / f'(x_{n}) ]
    '''
    if debug:
        print("=== ex1.py: getNewotonRaphsonRoot()")
    nSteps = 0
    roots  = [x]
    fVal   = f(x)

    while abs(f(x)) > epsilon and nSteps < nStepsMax:
        try:
            x = x - f(x) / dfdx(x)
        except:
            print("\tError! - derivative zero for x = ", x)
            exit(1)
        roots.append(x)
        fVal = f(x)
        nSteps+=1
        if debug:
            print("\tStep %d, x=%.6f, epsilon=%e" % (nSteps, roots[-1], fVal) )

    # Either a solution is found, or too many iterations
    if abs(fVal) > epsilon:
        nSteps = -1
    return roots, nSteps

def plotRoot(x, y, c="k", index=0):

    plt.plot([x], [0], c+"o")
    plt.plot([x, x], [0, y], c+':')
    plt.text(x-0.12, -0.25, r"$\left(x_{%d}, 0\right)$" % (index), color=c)

    plt.plot([x], [y], c+"o", fillstyle='none')
    plt.text(x-0.12, y+0.25, r"$\left(x_{%d}, f(x_{%d})\right)$" % (index,
    index), color=c)
```

```python
    return


# Define symbolic variable and expression
x  = s.Symbol("x")
F  = x**3 - 2*x**2 + 2
DF = s.diff(F, x)

# Create Python function f and df from SymPy expressions F and DF (evaluated numerically)
f   = s.lambdify(x, F, 'numpy')
df  = s.lambdify(x, DF, 'numpy')
eq  = s.Eq(F, 0)
exactRoots = s.solve(eq, x)
realRoots  = [root.evalf() for root in exactRoots if root.is_real]

print("=== Function and derivative:\n\tf(x) = ", F)
print("\tdf(x) = ", DF)

# Define various variables
xSeed    = 1.5
epsilon  = 1e-06
nSteps   = 3      # number of iterations
rootList = [xSeed]  # first root is the seed
xList    = np.arange(-1.0, 3.5, 0.0001) # np.linspace(-0.5, 2.3, 400)
fxList   = f(xList)
cList    = ["k", "g", "m", "c", "y"]
saveName =  "ex1_x0%s" % ( str(xSeed).replace(".", "p") )
np_Roots, np_Steps = getNewtonRaphsonRoot(f, df, xSeed, epsilon)
np_X     = np_Roots[-1]
if np_Steps > 0:
    print( "=== ex1.py\n\tFound root to be '%.3f' after %d iterations!" % (np_X, np_Steps) )
else:
    print("=== ex1.py\n\tSolution not found!")


print("=== ex1.py: Plot")
# Create a figure with subplots
plt.figure(figsize=(16, 6)).suptitle("Newton-Raphson Method")
#plt.figure(figsize=(16, 6)).suptitle(r"$f(x) = %s$   ,   $f\,'(x) = %s$" % (s.latex(F), s.latex(DF)) )

# Perform Newton-Raphson iterations
for i in range(nSteps):

    # Coordinates of the root
    x0 = rootList[-1]

    # Construct the tangent equation y_{tan} = f(x0) + f'(x0)*(x-x0)
    tList = [f(x0) + df(x0) * (x - x0) for x in xList]

    # Find the next root using Newton-Raphson
```

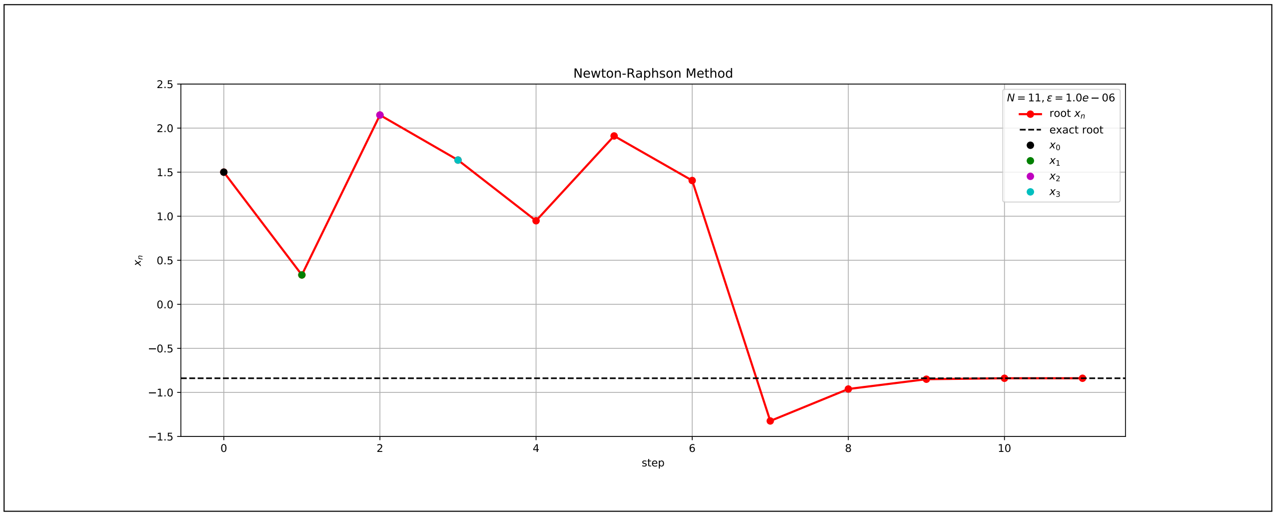
```python
    xi = x0 - f(x0) / df(x0)
    rootList.append(xi)
    print("\tStep %d, x_{%d}=%.3f, x_{%d} = %.3f - f(%.3f) / df(%.3f) = %.3f" % (i+1, i, x0, i+1, x0, x0, x0, xi) )

    # Create each plot on a separate pad
    plt.subplot(1, nSteps, i+1)

    # Plot the function we are trying to find the root of
    plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=%s$' % s.latex(F) )

    # Plot the current root
    plotRoot(x0, f(x0), cList[i], i)

    # Plot the tangent point/line
    plt.plot(xList, tList, "r--", label=r"slope $f\,'(x_{%d})$" % (i) )

    # Add y=0
    plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")

    # Plot the next root estimate
    plotRoot(xi, f(xi), cList[i+1], i+1)

    # Customise axes
    plt.title(f'Step {i+1}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.ylim(-1, 6)
    plt.legend(loc='upper left')
    plt.grid(False)
plt.tight_layout()
for ext in [".png", ".pdf"]:
    plt.savefig(saveName + ext)

# Create a figure with the roots
plt.figure(figsize=(16, 6))
plt.plot(np_Roots, "ro-", lw=2, label=r'root $x_{n}$')
plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")
for i in range(nSteps+1):
    plt.plot([i], np_Roots[i], cList[i]+"o", label=r'$x_{%d}$' % (i))
plt.xlabel('step')
plt.ylabel(r'$x_{n}$')
plt.ylim(-1.5, +2.5)
plt.title('Newton-Raphson Method')
plt.grid(True)
plt.legend(loc='upper right', title=r'$N=%d, \epsilon=%.1e$' % (np_Steps, epsilon) )
for ext in [".png", ".pdf"]:
    plt.savefig(saveName + "-roots" + ext)
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

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex3.py
   python3 ex3.py
   script -q ex3.log python3 -i ex3.py
'''
import sympy as s
import numpy as np
import matplotlib.pyplot as plt

def secant(f, x0, x1):
    x = None
    try:
        num = f(x1) * (x1 - x0)
        den = f(x1) - f(x0)
        x = x1 - num/den
    except:
        print("\tError! Denominator is zero for x0 = %s, x1 = %s" % (x0, x1) )
        exit(1)
    return x

def getSecantRoot(f, x0, x1, epsilon, nStepsMax=100, debug=True):
    '''
    x_{n+1} = x_{n} - f(x_{n}) [ (x_{n} - x_{n-1}) / ( f(x_{n}) - f(x_{n-1}) ) ]
    '''
    if debug:
        print("=== ex2.py: getNewotonRaphsonRoot()")
    nSteps = 0
    roots  = [x0, x1]
    fVal0  = f(x0)
    fVal1  = f(x1)

    while abs(fVal1) > epsilon and nSteps < nStepsMax:
        try:
            num = fVal1 * (x1 - x0)
            den = fVal1 - fVal0
            x = x1 - num/den
        except:
            print("\tError! Denominator is zero for x0 = %s, x1 = %s" % (x0, x1) )
            exit(1)
        roots.append(x)

        if debug:
            print("\tStep %d, x0=%.6f, x1=%.6f, x2=%.6f, epsilon=%e" % (nSteps, x0, x1, x, fVal1) )
        # Re-set values
```

```python
        x0 = x1
        x1 = x
        fVal0 = fVal1
        fVal1 = f(x1)
        nSteps+=1

    # Either a solution is found, or too many iterations
    if abs(fVal1) > epsilon:
        nSteps = -1
    return roots, nSteps

def plotRoot(x, y, c="k", index=0):

    plt.plot([x], [0], c+"o")
    plt.plot([x, x], [0, y], c+':')
    plt.text(x-0.12, -0.25, r"$\left(x_{%d}, 0\right)$" % (index), color=c)

    plt.plot([x], [y], c+"o", fillstyle='none')
    plt.text(x-0.12, y+0.25, r"$\left(x_{%d}, f(x_{%d})\right)$" % (index, index), color=c)
    return


# Define symbolic variable and expression
x  = s.Symbol("x")
F  = x**3 - 2*x**2 + 2

# Create Python function f and df from SymPy expression F
f  = s.lambdify(x, F, 'numpy')
eq = s.Eq(F, 0)
exactRoots = s.solve(eq, x)
realRoots  = [root.evalf() for root in exactRoots if root.is_real]

# Define various variables
x0       = 2.0 # 2.2 # 2.7 #2.0
x1       = 1.3 # 1.8 # x0-1.0 #x0-0.8
epsilon  = 1e-06
nSteps   = 3
rootList = [x0, x1]  # new two seeds
xList    = np.arange(-1.0, 3.5, 0.0001)
fxList   = f(xList)
cList    = ["k", "g", "m", "c", "y"]
saveName =  "ex2_x0%s_x1%s" % ( str(x0).replace(".", "p"), str(x1).replace(".", "p") )
sec_Roots, sec_Steps = getSecantRoot(f, x0,  x1, epsilon)
sec_X      = sec_Roots[-1]
if sec_Steps > 0:
    print( "=== ex2.py\n\tFound root to be '%.3f' after %d iterations!" % (sec_X, sec_Steps) )
else:
    print("=== ex2.py\n\tSolution not found!")
```

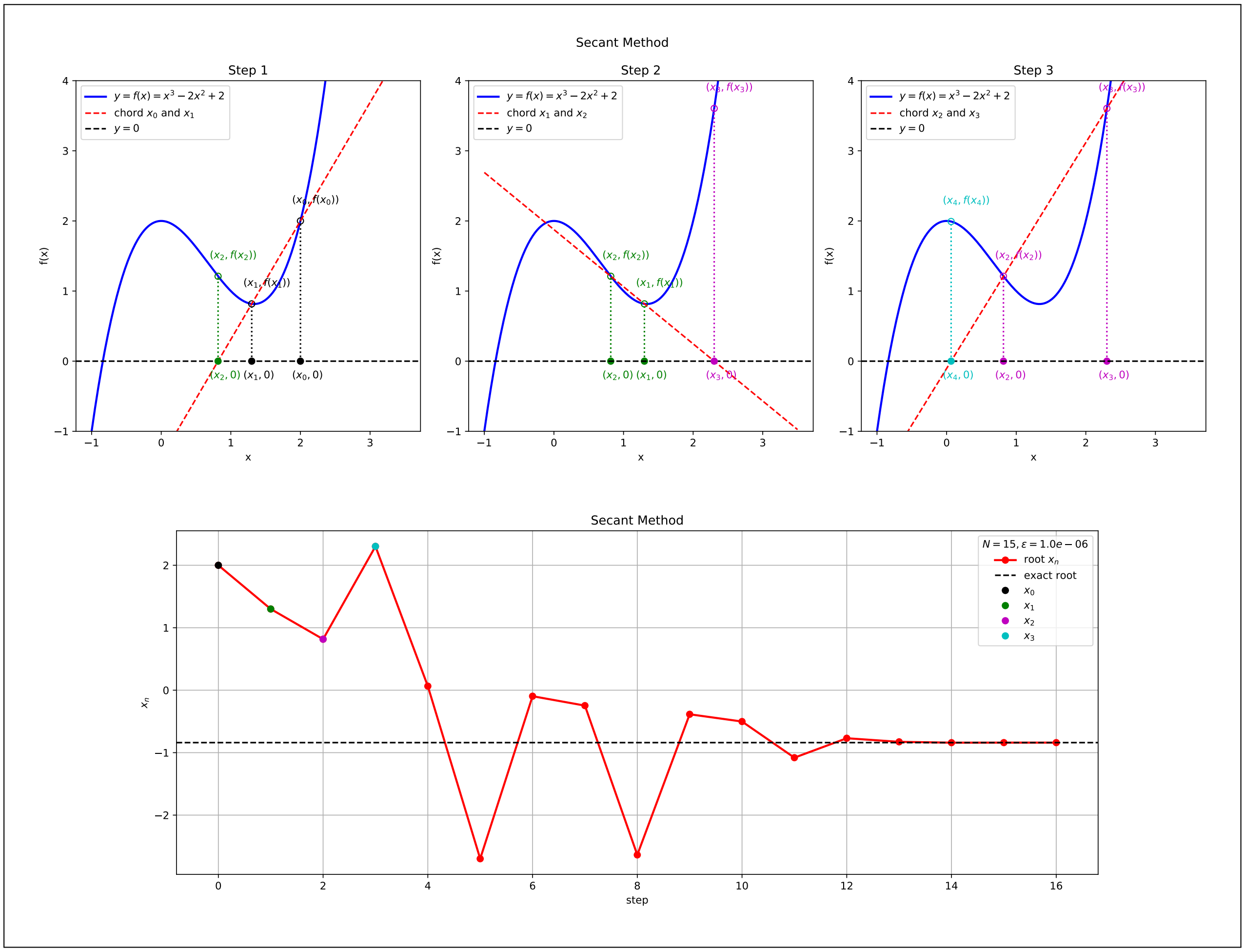
```python
print("=== ex2.py: Plot")
# Create a figure with subplots
plt.figure(figsize=(16, 6)).suptitle("Secant Method")

# Perform iterations
for i in range(0, nSteps, 1):

    # Coordinates of the root
    x0 = rootList[i]
    x1 = rootList[i+1]
    x2 = secant(f, x0, x1) # find the next root using Secant method
    rootList.append(x2)

    # Construct the tangent equation y_{tan} = f(x0) + f'(x0)*(x-x0)
    tList = [f(x0) + (f(x1) - f(x0)) * (x - x0) / (x1 - x0) for x in xList]
    print("\tStep %d, x_{%d}=%.2f, x_{%d} = %.2f, x_{%d} = %.2f" % (i+1, i, x0, i+1, x1, i+2, x2) )

    # Create each plot on a separate pad
    plt.subplot(1, nSteps, i+1)

    # Plot the function we are trying to find the root of
    plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=%s$' % (s.latex(F)) )

    # Plot the current root
    plotRoot(x0, f(x0), cList[i], i)
    plotRoot(x1, f(x1), cList[i], i+1)

    # Plot the tangent point/line
    plt.plot(xList, tList, "r--", label=r"chord $x_{%d}$ and $x_{%d}$" % (i, i+1) )

    # Add y=0
    plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")

    # Plot the next root estimate
    plotRoot(x2, f(x2), cList[i+1], i+2)

    # Customise axes
    plt.title(f'Step {i+1}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.ylim(-1, 4)
    plt.legend(loc='upper left')
    plt.grid(False)
plt.tight_layout()
for ext in [".png", ".pdf"]:
    plt.savefig(saveName + ext)

# Create a figure with the roots
plt.figure(figsize=(16, 6))
plt.plot(sec_Roots, "ro-", lw=2, label=r'root $x_{n}$')
plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")
```

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

Αποτέλεσμα:

```
=== ex2.py: getNewotonRaphsonRoot()
  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

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex3.py
   python3 ex3.py
   script -q ex3.log python3 -i ex3.py
'''
import sympy as s

# Define symbolic variables
x, y = s.symbols('x y')


# Function definitions
f  = s.sin(x)/x
print("Function f(x):\n\t", f)


# Differentiation / Integration wrt x
df_dx = s.diff(f , x)
intx_f = s.integrate(df_dx, x)
print("Differentiation df(x)/dx:\n\t", df_dx)
print("Integral of df_dx:\n\t", intx_f)


# Limit evaluation (in SymPy the symbol s.oo represent positive infinity)
lim_x_0   = s.limit(f, x, 0)        # Limit of f as x approaches 2
lim_x_inf = s.limit(f, x, s.oo)   # Limit of 1/f as x approaches infinity
print("Limit as x -> 0:\n\t"    , lim_x_0)
print("Limit as x -> infty:\n\t", lim_x_inf)
# L'Hopital rule: lim_{x->c} f(x) / g(x) == lim_{x->c} f'(x) / g'(x)


# Equation solving
eq = s.Eq(f, 0)  # Define the equation f = 0
solutions = s.solve(eq, x)  # Solve for x in the equation
print("Solutions of f(x) = 0:\n\t", solutions)


# Taylor expansion around x=0, up to 5 terms
taylor_expansion_f = s.series(f, x, 0, 5)
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

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex3.py
   python3 ex3.py
   script -q ex3.log python3 -i ex3.py
'''
import sympy as s
import numpy as np
import matplotlib.pyplot as plt

def bisection(f, xL, xR):
    fxL = f(xL)
    fxR = f(xR)
    x   = (xL + xR) / 2
    fx  = f(x)

    if (fx * fxL > 0):
        xL  = x
        fxL = f(xL)
    else:
        xR = x

    # Find the new midpoint
    x  = (xL + xR) / 2
    fx = f(x)
    return x, xL, xR


def getBisectionRoots(f, xL, xR, epsilon, nStepsMax=100, debug=True):
    '''
    Finds the root of the function f within the interval [xL, xR] using the Bisection method.
    '''
    if debug:
        print("=== ex4.py")

    fxL = f(xL)
    fxR = f(xR)

    if fxL * fxR > 0:
        print("Error! Function does not have opposite signs at interval endpoints [!")
        exit(1)

    nSteps = 1
    roots  = []
    x      = (xL + xR) / 2
    fx     = f(x)

```

```
    while abs(fx) > epsilon and nSteps < nStepsMax:

        if (fx * fxL > 0):
            # same sign
            xL  = x
            fxL = f(xL)
        else:
            # opposite sign
            xR = x

        # Find the new midpoint
        x  = (xL + xR) / 2
        fx = f(x)
        roots.append(x)
        if debug:
            print("\tStep %d, xL=%.6f, xR=%.6f, epsilon=%e" % (nSteps, xL, xR, (xR - xL) / 2))
        nSteps += 1

    return roots, nSteps

def plotRoot(x, y, c, subscript):

    plt.plot([x], [0], c+"o")
    plt.plot([x, x], [0, y], c+':')
    plt.text(x-0.12, +0.25, r"$\left(x_{%s}, 0\right)$" % (subscript), color=c)

    plt.plot([x], [y], c+"o", fillstyle='none')
    plt.text(x-0.12, y+0.25, r"$\left(x_{%s}, f(x_{%s})\right)$" % (subscript, subscript), color=c)
    return


# Define symbolic variable and expression
x  = s.Symbol("x")
F  = x**3- 2*x**2 + 2
f  = s.lambdify(x, F, 'numpy')
eq = s.Eq(F, 0)
exactRoots = s.solve(eq, x)
realRoots  = [root.evalf() for root in exactRoots if root.is_real]


# Define various variables
xL        = -1.3
xR        = +1.3
x         = (xL + xR) / 2
epsilon   = 1e-06
nSteps    = 3
rootList = []
xList     = np.arange(xL, xR, 0.0001)
fxList    = f(xList)
cList     = ["k", "g", "m", "c", "y"]
bi_Roots, bi_Steps = getBisectionRoots(f, xL, xR, epsilon)
```

```python
# x, xL, xR = bisection(f, xL, xR)
if nSteps > 0:
    print( "=== ex4.py\n\tFound root to be '%.3f' after %d iterations!" % (
   bi_Roots[-1], bi_Steps) )
else:
    print("=== ex4.py\n\tSolution not found!")


print("=== ex4.py: Plot")
# Create a figure with subplots
plt.figure(figsize=(16, 6)).suptitle("Bisection Method")

# Perform Newton-Raphson iterations
for i in range(nSteps):

    # Create each plot on a separate pad
    plt.subplot(1, nSteps, i+1)

    # Plot the function we are trying to find the root of
    plt.plot(xList, fxList, "b-", lw=2, label=r'$y=f(x)=%s$' % s.latex(F))

    # Add y=0
    plt.axhline(0, color="k", linestyle="--", label=r"$y=0$")

    # Plot the current and next roots
    plotRoot(xL, f(xL), cList[i], "L")
    plotRoot(xR, f(xR), cList[i], "R")
    plotRoot(x, f(x), cList[i+1], "")
    rootList.append(x)
    print("\tStep %d/%d, xL=%.3f, xR = %.3f, x = %.3f" % (i+1, nSteps, xL, xR,
   x) )

    # Find the next root using Newton-Raphson
    x, xL, xR = bisection(f, xL, xR)

    # Customise axes
    plt.title(f'Step {i+1}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.ylim(-4.0, 2.5)
    plt.legend(loc='lower right')
    plt.grid(True)

plt.tight_layout()
for ext in [".png", ".pdf"]:
    plt.savefig("ex4" + ext)

# Create a figure with the roots
plt.figure(figsize=(16, 6))
plt.plot(bi_Roots, "ro-", lw=2, label=r'root $x_{n}$')
plt.axhline(realRoots[0], color="k", linestyle="--", label="exact root")
for i in range(nSteps+1):
    plt.plot([i], bi_Roots[i], cList[i]+"o", label=r'$x_{%d}$' % (i))
```

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

Παράδειγμα 4 συνεχίζεται. . .