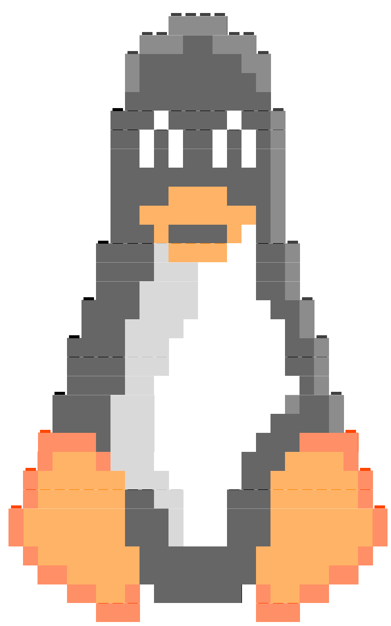
ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΚΑΙ ΕΦΑΡΜΟΣΜΕΝΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΦΥΣΙΚΗΣ

ΦΥΣ 140 Εισαγωγή στην Επιστημονική Χρήση Υπολογιστών
Χειμερινό Εξάμηνο 2023

Φώτης Πτωχός και Αλέξανδρος Αττίκης

Φροντιστήριο 10

14 Νοεμβρίου 2023
15:00 - 17:00



Φροντιστήριο 10

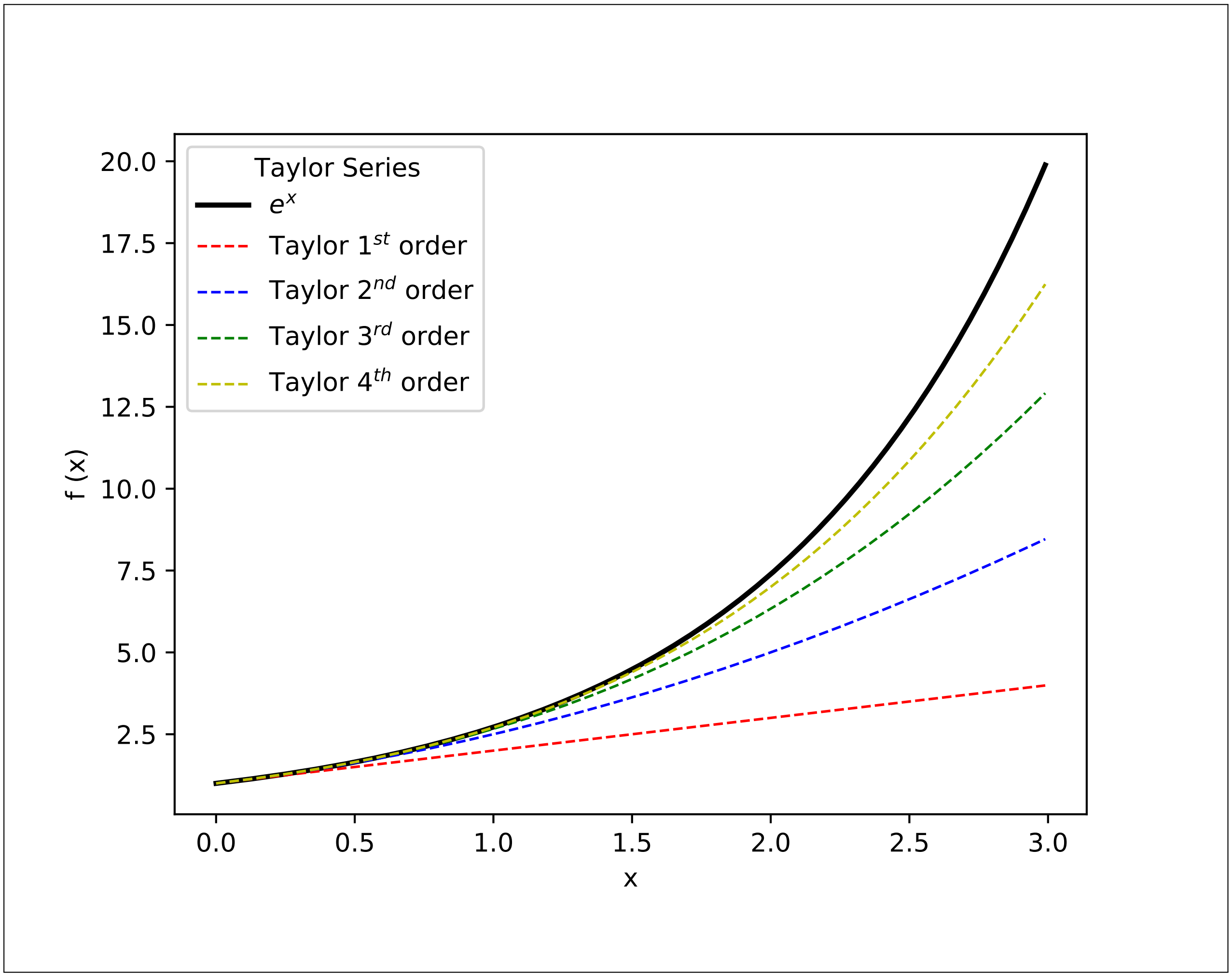
Παράδειγμα 1 Η εκθετική συνάρτηση e^x έχει σειρά Taylor που δίδεται από τη σχέση $e^x = \sum_{n=0}^{n=\infty} \frac{x^n}{n!}$.

Ας το δούμε και γραφικά όμως με το παρακάτω πρόγραμμα:

tutorial10/ex1.py

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex1.py
   python3 ex1.py
   script -q ex1.log python3 -i ex1.py


DESCRIPTION:
'''
import numpy as np
import matplotlib.pyplot as plt
import math as math

xList = [0.01*x for x in range(0, 300, 1)]
yList = [np.exp(x) for x in xList]
y1List= [1 + x for x in xList]
y2List= [1 + x + x**2/2 for x in xList]
y3List= [1 + x + x**2/2 + x**3/math.factorial(3) for x in xList]
y4List= [1 + x + x**2/2 + x**3/math.factorial(3) + x**4/math.factorial(4) for x in xList]

plt.figure()
plt.plot(xList, yList , 'k-' , lw=2, label= r"$e^{x}$")
plt.plot(xList, y1List, 'r--', lw=1, label= r"Taylor $1^{st}$ order")
plt.plot(xList, y2List, 'b--', lw=1, label= r"Taylor $2^{nd}$ order")
plt.plot(xList, y3List, 'g--', lw=1, label= r"Taylor $3^{rd}$ order")
plt.plot(xList, y4List, 'y--', lw=1, label= r"Taylor $4^{th}$ order")
plt.xlabel('x')
plt.ylabel('f (x)')
plt.legend(title="Taylor Series")
for ext in [".png", ".pdf"]:
    plt.savefig(__file__.split(".")[0] + ext)
plt.show()
```

Αποτέλεσμα:


Taylor Series
e^x
Taylor 1^{st} order
Taylor 2^{nd} order
Taylor 3^{rd} order
Taylor 4^{th} order
f (x)
x
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

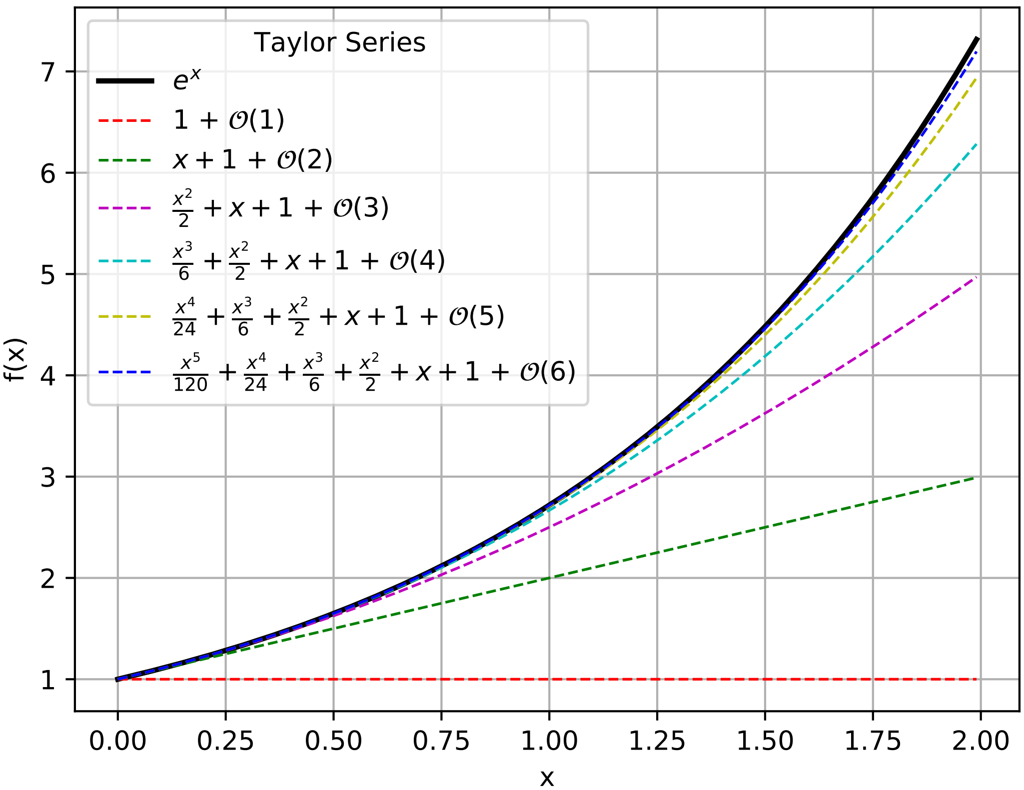
Παράδειγμα 2 Θα μπορούσαμε να δημιουργήσουμε την γραφική αναπαράσταση της σειρά Taylor για την εκθετική συνάρτηση e^x έ με τη χρήση της βιβλιοθήκης *sympy*, όπως παρακάτω:

tutorial10/ex2.py

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex2.py
   python3 ex2.py
   script -q ex2.log python3 -i ex2.py


DESCRIPTION:
'''
import numpy as np
import matplotlib.pyplot as plt
import sympy as s


# Use sympy to a symbolic variable (x) and the expression
x    = s.Symbol('x')
F    = s.exp(x)
f    = s.lambdify(x, F)
oMin = 1
oMax = 6

# Create a list of x values
cList = ["r", "g", "m", "c", "y", "b"]
xList = np.arange(0.0, 2, 0.01)
yList = [f(xVal) for xVal in xList]
tList = []
for o in range(oMin, oMax+1):

        # Generates the Taylor series expansion of the function F wrt variable x,
        # centered at x=0, and up to the specified order 'o'
        tSeries = s.series(F, x, 0, o)

        # Truncate the series, keeping only the terms up to the specified order 'o'
        tSeries = tSeries.removeO()

        tList+= [tSeries]
        print("Order %d) taylor = %s" % (o, tSeries))

# Plotting
plt.figure()
plt.plot(xList, yList, 'k-', lw=2, label=r"$%s$" % (s.latex(F) ) )
for i, t in enumerate(tList, oMin):
    plt.plot(xList, [t.subs(x, xVal) for xVal in xList], cList[i-oMin]+'--', lw=1, label=r"$%s$ + $\mathcal{O}(%d)$" % (s.latex(tList[i-oMin]), i) )
plt.xlabel('x')
plt.ylabel('f(x)')
```

```python
plt.legend(title="Taylor Series")
plt.grid(True)
for ext in [".png", ".pdf"]:
    plt.savefig(__file__.split(".")[0] + ext)
plt.show()
```

Αποτέλεσμα:



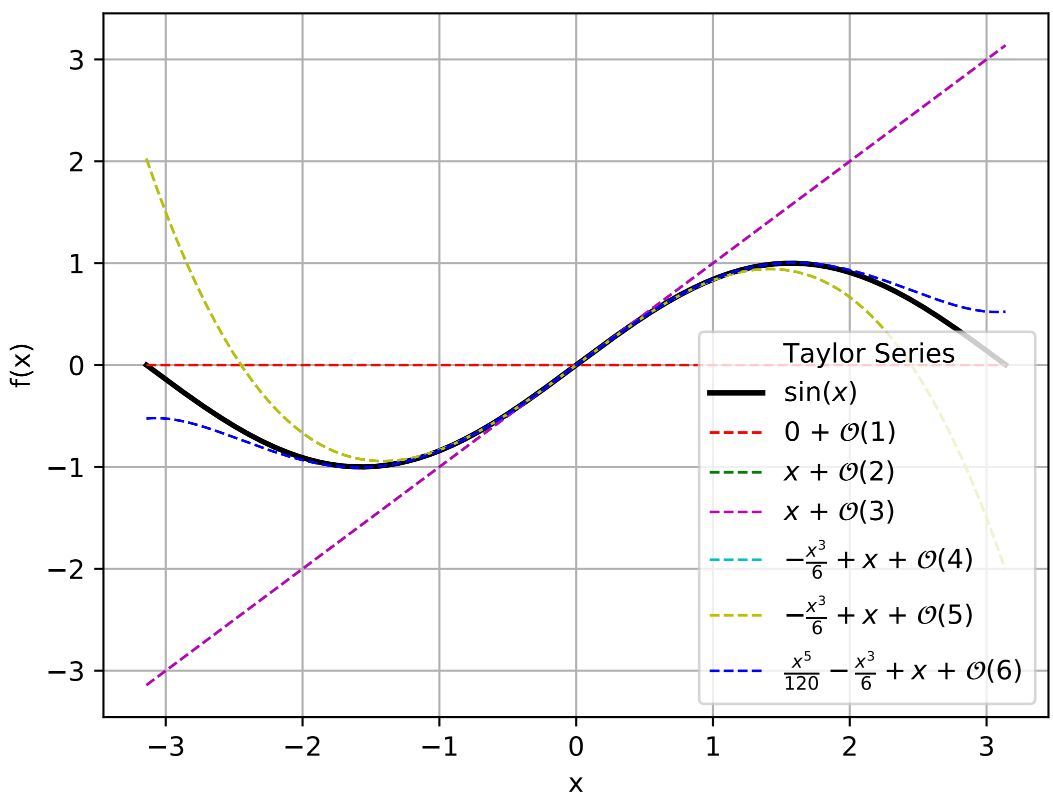
Παράδειγμα 3 Ο κώδικας μας μπορεί να χρησιμοποιηθεί αυτούσιως για τη μελέτη οποιασδήποτε συνάρτησης, αλλάζοντας μόνο τη μεταβλητή F. Σε αυτό το παράδειγμα μελετούμε το ανάπτυγμα Taylor για τη συνάρτηση $\sin x$:

tutorial10/ex3.py

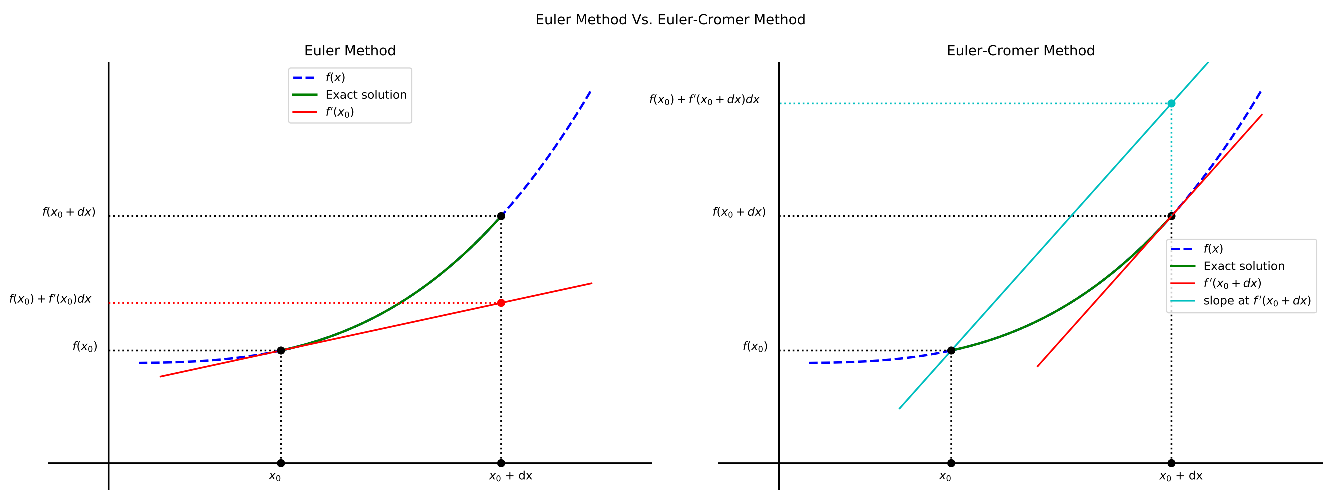
```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex2.py
   python3 ex2.py
   script -q ex2.log python3 -i ex2.py


DESCRIPTION:
'''
import numpy as np
import matplotlib.pyplot as plt
import sympy as s


# Use sympy to a symbolic variable (x) and the expression
x    = s.Symbol('x')
F    = s.sin(x)
f    = s.lambdify(x, F)
oMin = 1
oMax = 6

# Create a list of x values
cList = ["r", "g", "m", "c", "y", "b"]
xList = np.arange(-1*np.pi, 1*np.pi, 0.01)
yList = [f(xVal) for xVal in xList]
tList = []
for o in range(oMin, oMax+1):

        # Generates the Taylor series expansion of the function F wrt variable x,
        # centered at x=0, and up to the specified order 'o'
        tSeries = s.series(F, x, 0, o)

        # Truncate the series, keeping only the terms up to the specified order 'o'
        tSeries = tSeries.removeO()

        tList+= [tSeries]
        print("Order %d) taylor = %s" % (o, tSeries))

# Plotting
plt.figure()
plt.plot(xList, yList, 'k-', lw=2, label=r"$%s$" % (s.latex(F) ) )
for i, t in enumerate(tList, oMin):
    plt.plot(xList, [t.subs(x, xVal) for xVal in xList], cList[i-oMin]+'--', lw=1, label=r"$%s$ + $\mathcal{O}(%d)$" % (s.latex(tList[i-oMin]), i) )
plt.xlabel('x')
```

```
plt.ylabel('f(x)')
plt.legend(title="Taylor Series")
plt.grid(True)
for ext in [".png", ".pdf"]:
    plt.savefig(__file__.split(".")[0] + ext)
plt.show()
```

Αποτέλεσμα:



Παράδειγμα 4 Στη Διάλεξη 10 είδαμε πως μπορούμε να περιγράψουμε ένα σώμα σε ελεύθερη πτώση με τη μέθοδο Euler αλλά και τη μέθοδο Euler-Cromer. Στο απλό αυτό παράδειγμα, που αφορά τη κίνηση ενός σώματος σε ένα σταθερό πεδίο βαρύτητας ('βαλλιστική κίνηση'), ξεκινάμε με μια αρχική ταχύτητα (και άρα ενέργεια) και έπειτα αφήνουμε το σώμα να κινηθεί ελεύθερα. Εδώ η κατακόρυφη επιτάχυνση είναι σταθερή, έτσι ώστε $\ddot{y} = -9.81\,ms^{-2}$ (η επιτάχυνση είναι σταθερή και καθοδική), με αρχικές συνθήκες $y(t=0) = y_0 = 0.0\,m$ (εκκίνηση από το έδαφος) και αρχική ταχύτητα $\dot{y}(t=0) = v_0 = 5\,ms^{-1}$. Αγνοούμε την οριζόντια κίνηση, που είναι ανεξάρτητη από την κατακόρυφη κίνηση, αλλά και αρκετά εύκολο να περιγραφεί αφού δεν υπάρχει οριζόντια επιτάχυνση. Με τη μέθοδο του Euler, αυτό που πρέπει να κάνουμε είναι να ακολουθήσουμε την εφαπτομένη της καμπύλης της λύσης για ένα διάστημα dt. Κατόπιν, υπολογίζουμε και πάλι την κλίση της καμπύλης στο νέο σημείο και προχωρούμε με βάση τη κλίση αυτή στο επόμενο διάστημα, και ούτω καθεξής. Με τη μέθοδο του Euler-Cromer, ο αλγόριθμος που πρέπει να ακολουθήσουμε για να εξελίξουμε χρονικά το σύστημα μας είναι σχεδόν ακριβώς ο ίδιος. Η μόνη διαφορά είναι πως, σε αυτή τη περίπτωση, πρώτα υπολογίζεται η κλίση της καμπύλης (παράγωγος της θέσης συναρτήσεις του χρόνου) στη δεδομένη χρονική στιγμή t και κατόπιν χρησιμοποιείται η παράγωγος αυτή για τον υπολογισμό της θέσης του σώματος. Το γεγονός αυτό μας προτρέπει να χρησιμοποιήσουμε μόνο μία συνάρτηση για την εύρεση και επιστροφή των προσεγγιστικών μας λύσεων, η οποία χρησιμοποιεί τη μεταβλητή method ως όρισμα για να διακρίνει ποια από τις δύο μεθόδους, Euler ή Euler-Cromerθα χρησιμοποιήσει.



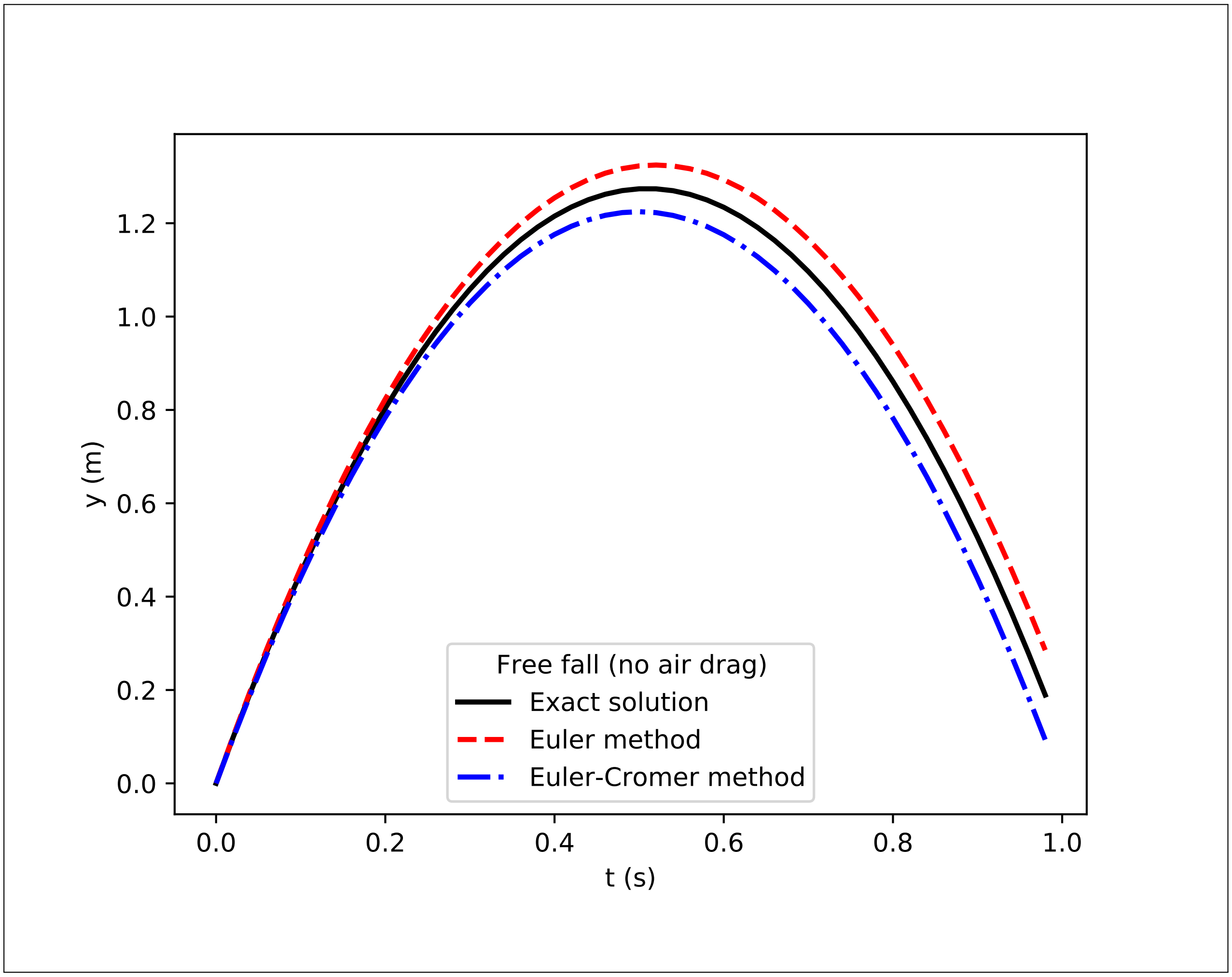
tutorial10/ex4.py

```
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex4.py
   python3 ex4.py
   script -q ex4.log python3 -i ex4.py


```

```python
DESCRIPTION:
'''
import numpy as np
import matplotlib.pyplot as plt

def solveFreeFall(y0, dydt, v0, dvdt, dt, tMax, method):
    '''
    Solve a first-order ordinary differential equation using Euler or Euler-Cromer method.

    Parameters:
    - y0 ......: Initial position along y-axis
    - dydt ....: Function representing the differential equation f(x, t) = dy/dt
    - v0 ......: Initial velocity
    - dvdt ....: Function representing the differential equation f(v, t) = dv/dt
    - dt ......: Time step to be used
    - tMax ....: Total time our system is being investigaed
    - method ..: "euler" or "euler-cromer"

    Returns:
    - xList: List of time values over time
    - yList: List of approximate position values over time
    - tList: List of exact position values over time
    '''

    # Define the lists to hold our values
    xList = []
    yList = []
    tList = []

    # Define the variables that will be used to describe time lapsed and intantaneous velocity
    t = 0
    y = y0
    v = v0

    # Time evolution of the system until t = tMax for nSteps = tMax/dt
    while t < tMax:

        # Save time, exact solution, and approximate solution
        xList.append(t)
        tList.append(v0*t - g*t*t/2.0) # equations of motion (Newton)
        yList.append(y)

        # Euler or Euler-Crome step for position and velocity (approximate solution
        if method == 'euler':
               y =    y + dydt * dt
            dydt = dydt + dvdt * dt
        elif method == 'euler-cromer':
            dydt = dydt + dvdt * dt
```

```python
                y =    y + dydt * dt
        else:
            print("Invalid method. Choose 'euler' or 'euler-cromer'.")
            exit()

        # Time evolution of our systems
        t = t + dt

    return xList, yList, tList



# Define variables
g    = 9.81   # acceleration [ms^{-2}]
dt   = 0.02   # time step [s]
y0   = 0.0    # initial position [m]
v0   = 5.0    # initial velocity [ms^{-1}]
dydt = v0     # velocity at t=0, time-dependent [ms^{-1}]
dvdt = -g     # acceleration at t=0, time-independent [ms^{-2]
tMax = 1.0    # the total time that our system will be evolved [s]

# Apply Euler or Euler-Cromer to find approximate solutions
x1List, y1List, t1List = solveFreeFall(y0, dydt, v0, dvdt, dt, tMax, "euler")
x2List, y2List, t2List = solveFreeFall(y0, dydt, v0, dvdt, dt, tMax, "euler-cromer")

# Plot the exact and approximate results and compare
plt.figure()
plt.plot(x1List, t1List, 'k-' , lw=2, label="Exact solution")
plt.plot(x1List, y1List, 'r--', lw=2, label="Euler method")
plt.plot(x1List, y2List, 'b-.', lw=2, label="Euler-Cromer method")
plt.xlabel('t (s)')
plt.ylabel('y (m)')
plt.legend(title="Free fall (no air drag)", loc="best")
for ext in [".png", ".pdf"]:
    plt.savefig(__file__.split(".")[0] + ext)
plt.show()
```

Αποτέλεσμα:


Free fall (no air drag)
Exact solution
Euler method
Euler-Cromer method
y (m)
t (s)
0.0
0.2
0.4
0.6
0.8
1.0
1.2

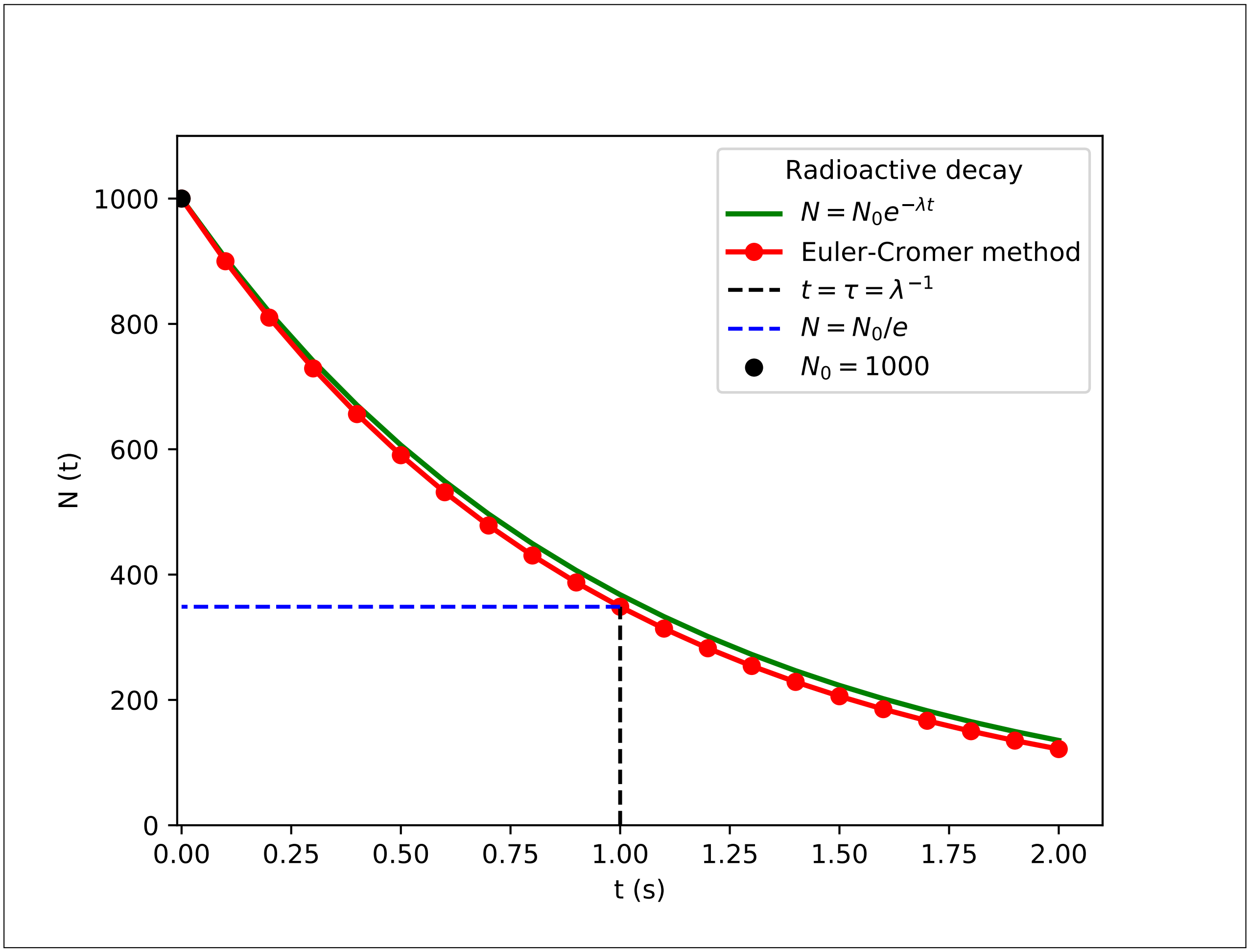
Παράδειγμα 5 Σε προηγούμενο μάθημα εξετάσαμε την πυρηνική διάσπαση $\frac{dN}{dt} = -\lambda N$. Αυτό είναι ένα από τα απλούστερα παραδείγματα διαφορικών εξισώσεων: μια πρώτης τάξης, γραμμική, διαχωρίσιμη διαφορική εξίσωση με μία εξαρτημένη μεταβλητή. Είδαμε ότι μπορούσαμε να μοντελοποιήσουμε τον αριθμό των ατόμων N βρίσκοντας μια αναλυτική λύση μέσω ολοκλήρωσης $N(t) = N_0 e^{-\lambda t}$. Αυτή η διαφορική εξίσωση μπορεί να λυθεί εάν χρησιμοποιήσουμε τη μέθοδο του Euler που είναι μια κατά προσέγγιση αριθμητική μέθοδος:

tutorial10/ex5.py

```python
#!/usr/bin/python3
'''
USAGE:
   chmod +x ex5.py
   python3 ex5.py
   script -q ex5.log python3 -i ex5.py


DESCRIPTION:
For N nuclei of a radioactive substance at time t, the decay rate
dN/dt is given by:

 tau dN/dt = -N
=> dN/dt = -N/tau = - lambda N
=> N (t) = No exp(-t/tau)
where No = N(t=0).

At the time of the decay rate (aka mean lifetime), i.e. t = tau,
the number of radioactive nuclei is about a third of the initial population:
N(t=tau) = No/exp(1) = No/0.37

Note that the decay constant  "lambda", is the reciprocal of the mean lifetime:
tau = 1/lambda
'''
import numpy as np
import matplotlib.pyplot as plt

def expDecay(N0, t, tau):
    return N0*np.exp(-t/tau)

def Euler(N, t, dt, tau, nSteps):
    NList = []
    tList = []

    for i in range(0, nSteps):

        NList.append(N)
        tList.append(t)

        # Evaluate the slope (derivative) at the start of the time interval (i.e. the decay rate!)
        dNdt = -N/tau

        # Evaluate the Nuclei after the time interval
```

```
        N = N + dNdt * dt

        # Increment time interval
        t = t + dt
    return tList, NList

# Define variables
t0     = 0.0  # seconds
tMax   = 2.0  # seconds
dt     = 0.1  # seconds
nSteps = int( (tMax - t0)/dt)+1
N0     = 1000 # initial number of nuclei population
tau    = 1.0  # lambda = 1/tau where tau is the decay constant

xList, yList = Euler(N0, t0, dt, tau, nSteps)
plt.figure()
plt.plot(xList, [expDecay(N0, t, tau) for t in xList], 'g-', lw=2, label= r"$N=
    N_{0}e^{-\lambda t}$")
plt.plot(xList, yList, 'ro-', lw=2, label= r"Euler-Cromer method")
plt.xlabel('t (s)')
plt.ylabel('N (t)')
plt.plot([tau, tau], [0, yList[int(tau/dt)] ], color='k',linestyle='--', label=
    r"$t=\tau=\lambda^{-1}$")
plt.plot([tau, 0], [yList[int(tau/dt)], yList[int(tau/dt)] ], color='b',
    linestyle='--', label=r"$N=N_{0}/e$")
plt.ylim(0, N0*1.1)
plt.xlim(-0.01, tMax+dt)
plt.plot([0], [N0], 'ko', label=r"$N_{0} = %0.0f$" % (N0) )
plt.legend(title="Radioactive decay")

# Save BEFORE you show the figure interactively
for ext in [".png", ".pdf"]:
    plt.savefig(__file__.split(".")[0] + ext)
plt.show()
```

Αποτέλεσμα:


Radioactive decay
$N = N_0 e^{-\lambda t}$
Euler-Cromer method
$t = \tau = \lambda^{-1}$
$N = N_0/e$
$N_0 = 1000$
N (t)
t (s)
0
200
400
600
800
1000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00