2η Άσκηση

Το πρόβλημα ρίψης σφαίρας από το έδαφος με αρχική ταχύτητα, λαμβάνοντας υπόψιν μόνο την δύναμη της βαρύτητας, μπορεί να διατυπωθεί μαθηματικά εφαρμόζοντας το δεύτερο νόμο του Newton, από τον οποίον προκύπτει το σύστημα διαφορικών εξισώσεων με αρχικές συνθήκες (1).

$$\begin{cases} m\frac{d^2x}{dt^2} = & 0 \\ m\frac{d^2y}{dt^2} = & -mg \\ x(0) = & 0 \\ y(0) = & 0 \\ x'(0) = & v_0 \cos\frac{\pi}{4} \\ y'(0) = & v_0 \sin\frac{\pi}{4} \end{cases} \tag{1}$$

Για να λυθεί το σύστημα με αριθμητικές μεθόδους, πρέπει να έρθει σε κατάλληλη μορφή $u' = f(t, u)$, όπου u διάνυσμα με άγνωστες συναρτήσεις, t η ανεξάρτητη μεταβλητή και f κατάλληλη συνάρτηση. Ορίζεται $u_0 = x$, $u_1 = x'$, $u_2 = y$ και $u_3 = y'$, οπότε οι εξισώσεις (1) μετασχηματίζονται σε μορφή πινάκων (2)

$$\underbrace{\frac{d}{dt}\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}}_{u'} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix}}_{u} + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ -g \end{bmatrix}}_{b}, \underbrace{\begin{bmatrix} 0 \\ v_0 \cos\frac{\pi}{4} \\ 0 \\ v_0 \sin\frac{\pi}{4} \end{bmatrix}}_{u(0)} \tag{2}$$

ή συνοπτικά (3), όπου αντιστοιχεί $f(t, u) = Au + b$. Σε αυτή τη μορφή το σύστημα διαφορικών εξισώσεων είναι επιλύσιμο με το σχήμα Euler ή Runge-Kutta 4ης τάξης.

$$u' = Au + b, u_0 = u(0) \tag{3}$$

Συγκεκριμένα, για το σχήμα Euler, εφαρμόζεται η υπολογιστική διαδικασία (4),

$$u_{i+1} = u_i + hf(t_i, u_i), u_0 = u(0) \tag{4}$$

ενώ στο σχήμα Runge-Kutta η (5).

$$\begin{aligned} k_1 =& f(t_i, u_i) \\ k_2 =& f(t_i + \frac{h}{2}, u_i + \frac{h}{2}k_1) \\ k_2 =& f(t_i + \frac{h}{2}, u_i + \frac{h}{2}k_2) \\ k_3 =& f(t_i + h, u_i + hk_3) \\ u_{i+1} =& u_i + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right), u_0 = u(0) \end{aligned} \tag{5}$$

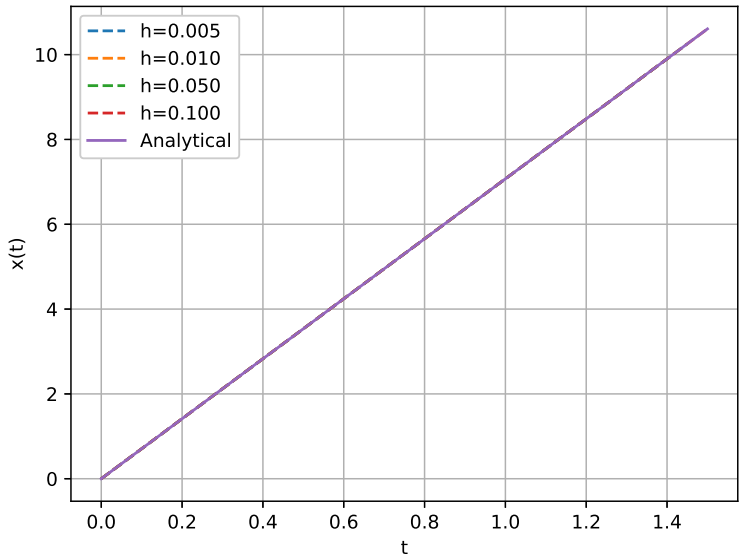
Η επίλυση του συστήματος διαφορικών εξισώσεων με τη μέθοδο Euler για διάφορα μεγέθη χρονικού βήματος παράγει τα Σχήματα 1 και 2. Μαζί με την αριθμητική λύση δίνεται και η αναλυτική που δίνεται από τις εξισώσεις 6.

$$\begin{cases} x(t) = & v_0 \cos(\phi)t \\ y(t) = & v_0 \sin(\phi)t - \frac{1}{2}gt^2 \end{cases} \tag{6}$$

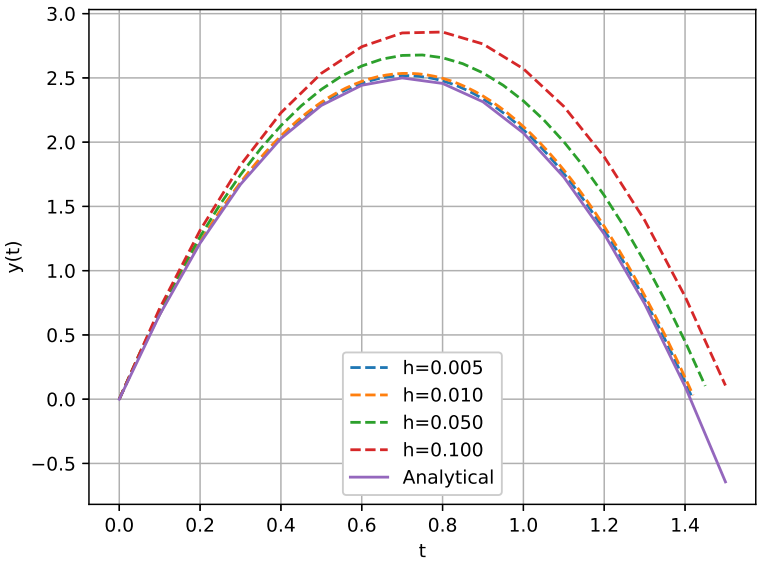
Η σύγκριση αποκαλύπτει ότι η αριθμητική λύση της $x(t)$ συμπίπτει με την αναλυτική για κάθε μέγεθος βήματος, που είναι αναμενόμενο γιατί η λύση είναι γραμμική συνάρτηση και συμπίπτει με την προσέγγιση πρώτης τάξης της μεθόδου Euler. Ωστόσο, η $y(t)$ που είναι πολυώνυμο 2ου βαθμού, απαιτεί αρκετά μικρό βήμα $h < 0.01$ προκειμένου να πλησιάσει ικανοποιητικά την αναλυτική. Μικρότερο βήμα συνεπάγεται μεγαλύτερο αριθμό βημάτων, δηλαδή 142 (και 283 για $h = 0.005$).

Η εφαρμογή της μεθόδου Runge-Kutta 4ης τάξης, παράγει καλύτερα αποτελέσματα. Και πάλι η αριθμητική λύση της $x(t)$ συμπίπτει με την αναλυτική, αλλά επιπλέον η αριθμητική λύση της $y(t)$ είναι πολύ κοντά στην αναλυτική ακόμα και για $h = 0.1$ δηλαδή μεγαλύτερο κατά μία τάξη μεγέθους σε σχέση με την Euler. Τα αποτελέσματα δίνονται στα Σχήματα 3 και 4. Ο αριθμός των βημάτων για κάθε μέγεθος βήματος είναι ίδιος, αλλά εφόσον η Runge-Kutta αποκλίνει πολύ λίγο από τη αναλυτική λύση για μεγάλο βήμα, αποδεικνύεται και η υπεροχή της ως προς την Euler.

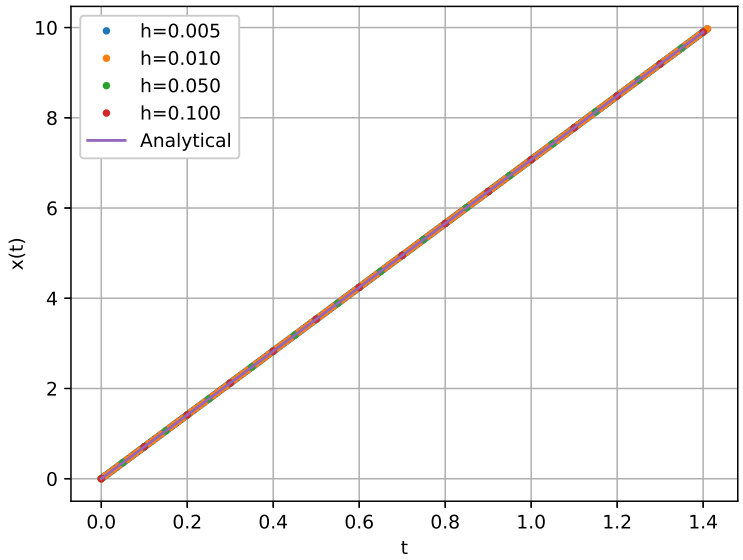
Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Ο Euler και Runge-Kutta 4ης τάξης δίνεται στο



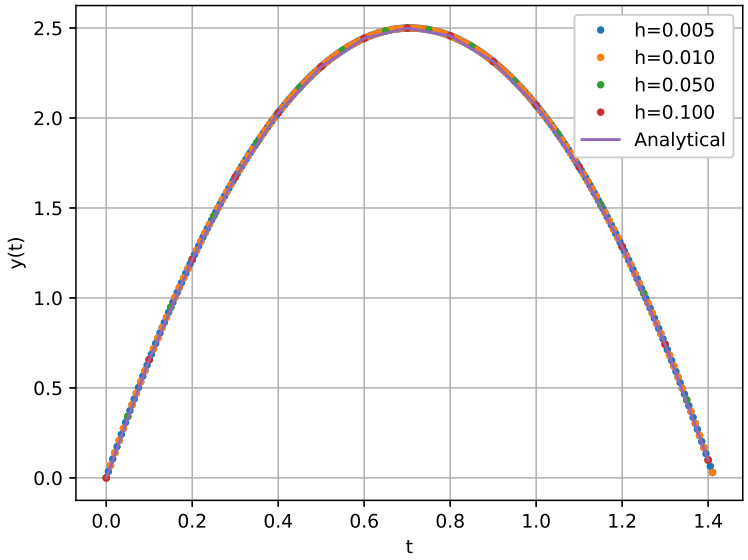
Σχήμα 1: Μέθοδος Euler - $x(t)$



Σχήμα 2: Μέθοδος Euler - $y(t)$



Σχήμα 3: Μέθοδος Runge-Kutta 4ης - $x(t)$



Σχήμα 4: Μέθοδος Runge-Kutta 4ης - $y(t)$

Παράρτημα.

ex2.py

```
import numpy as np
import matplotlib.pyplot as plt
import diffEquation as de


def f(t, y):
    g = 10.
    A = np.array([[0, 1, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 1],
                  [0, 0, 0, 0]])
    b = np.array([0, 0, 0, -g])
    return np.dot(A, y) + b


def run(Y0, f, method, h, figNum):
    t = np.array([])
    for hi in h:
        solution = de.solve(Y0, f, method, hi)
        print("h=%.3f, iterations=%d"% (hi,
                                        solution['iterations']))
        x = solution['x']
        y = solution['y']
        t = solution['t']
        plt.figure(figNum)
        plt.plot(t, x, '--', label="h=%.3f" % hi)
        plt.figure(figNum + 1)
        plt.plot(t, y, '--', label="h=%.3f" % hi)
    g = 10.
    v0x = Y0[1]
    v0y = Y0[3]
    x = v0x * t
    y = v0y * t - 1/2 * g * np.power(t, 2.)
    plt.figure(figNum)
    plt.plot(t, x, label="Analytical")
    plt.grid()
    plt.legend()
```

```
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.figure(figNum+1)
    plt.plot(t, y, label="Analytical")
    plt.grid()
    plt.legend()
    plt.xlabel('t')
    plt.ylabel('y(t)')


v0 = 10.  # Initial conditions
phi = np.pi / 4
v0x = v0 * np.cos(phi)
v0y = v0 * np.sin(phi)
Y0 = np.array([0, v0x, 0, v0y])  # initial vector
h = np.array([0.005, 0.01, 0.05, 0.1])  # time steps
t = np.array([])
plt.close('all')

print('Euler')
run(Y0, f, de.eulerStep, h, 1)  # run with Euler
print('\nRunge-Kutta 4th')
run(Y0, f, de.rungeKutta4, h, 3)  # run with Runge-Kutta 4th
```