3η Άσκηση

Η εύρεση της μέγιστης εμβέλειας με υπολογιστικό τρόπο απαιτεί καταρχάς την επίλυση του προβλήματος με αριθμητικούς τρόπους. Κατόπιν, εφόσον η λύση είναι αριθμητική, απαιτείται επίλυση του προβλήματος για ένα εύρος τιμών γωνίας ρίψης και εύρεση της μέγιστης εμβέλειας για κάθε γωνία, $x(\phi)$.

Το σύστημα διαφορικών εξισώσεων που περιγράφει τη βολή σφαίρας σε ομογενές πεδίο βαρύτητας με αεροδυναμική τριβή ανάλογη της ταχύτητας, δίνεται από τις εξισώσεις (1), όπου $\frac{C_D}{m} = \frac{g}{4}$.

$$\begin{cases} x'' = & \frac{C_D}{m} x' \\ y'' = & \frac{C_D}{m} y' - g \\ x(0) = & 0 \\ y(0) = & 0 \\ x'(0) = & v_0 \cos\phi \\ y'(0) = & v_0 \sin\phi \end{cases} \tag{1}$$

Ορίζεται $u_0 = x$, $u_1 = x'$, $u_2 = y$ και $u_3 = y'$, οπότε οι εξισώσεις (1) μετασχηματίζονται σε μορφή πινάκων (2)

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

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

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

Όπως προαναφέρθηκε, η λύση του προβλήματος ρίψης μπορεί να βρεθεί υπολογιστικά για διαφορετικές γωνίες $\phi \in [0, \frac{\pi}{4}]$. Μια απλή μέθοδος εύρεσης της μέγιστης εμβέλειας είναι η επίλυση του παραπάνω προβλήματος για διακριτές τιμές γωνίας και η εύρεση της μέγιστης εμβέλειας από τις τιμές που προέκυψαν.

Μια άλλη μέθοδος και πιο ακριβής είναι η θεώρηση ότι η συνάρτηση $R(\phi)$ είναι μίας μεταβλητής, συνεχής στο $[0, \frac{\pi}{4}]$. Από θεώρημα της Ανάλυσης αυτή θα λαμβάνει μια μέγιστη και μια ελάχιστη τιμή σε αυτό το κλειστό διάστημα. Το μέγιστο της συνάρτησης μπορεί να υπολογιστεί και πάλι αριθμητικά χρησιμοποιώντας κάποιον αλγόριθμο αριθμητικής ανάλυσης, όπως ο αλγόριθμος του Brent.

Ο λόγος που η δεύτερη μέθοδος είναι πιο ακριβής, είναι ότι ο κανόνας του Brent έχει σαν κριτήριο τερματισμού την επίτευξης μιας συγκεκριμένης ακρίβειας (πχ 10^{-3}). Επιπλέον, ο απλός δειγματολειπτικός αλγόριθμος θα επιλύσει το σύστημα διαφορικών εξισώσεων για διάφορες τιμές του ϕ, οι περισσότερες από τις οποίες είναι πολύ μακριά από το ϕ_{max} και επομένως δεν είναι πολύ αποδοτικός.

Για λόγους σύγκρισης υλοποιήθηκαν και οι δύο τρόποι και έδωσαν αποτελέσματα $\phi_{max}^{Br} = 27.61°$ με $R(\phi_{max}^{Br}) = 3.0647$ και $\phi_{max}^{Smpl} = 25.78°$ με $R(\phi_{max}^{Smpl}) = 3.0631$. Όπως προαναφέρθηκε, ο αλγόριθμος του Brent δίνει πιο ακριβές αποτέλεσμα.

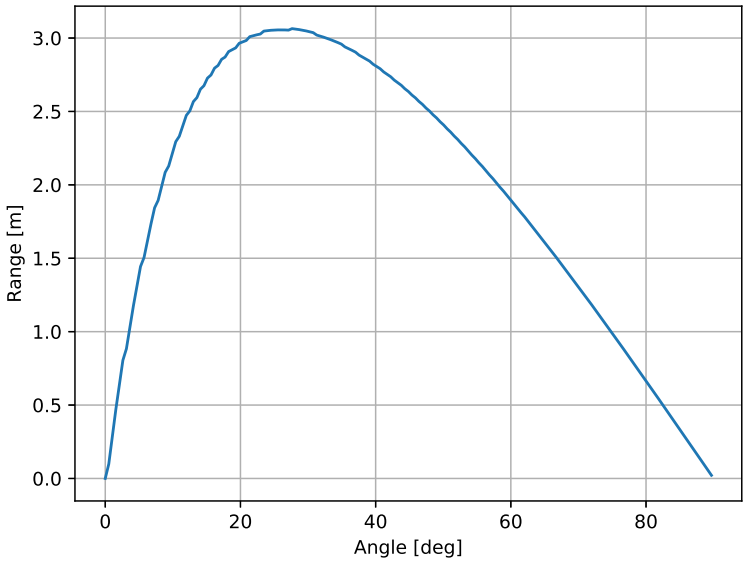
Τυπικά, ο αλγόριθμος του Brent υπολογίζει το ελάχιστο μιας συνάρτησης f, αλλά το ελάχιστο της f είναι το μέγιστο της $-f$ και αντίστροφα. Η επίλυση του συστήματος διαφορικών εξισώσεων έγινε με χρονικό βήμα $h = 0.02$, που είναι μια καλή τιμή για επίτευξη ομαλούς λύσης και ταχύτητας.

Στο Σχήμα 1 δίνεται η συνάρτηση της εμβέλειας που υπολογίστηκε αριθμητικά. Στο Σχήμα 2 δίνεται η τροχιά του βλήματος για την γωνία μέγιστης εμβέλειας.

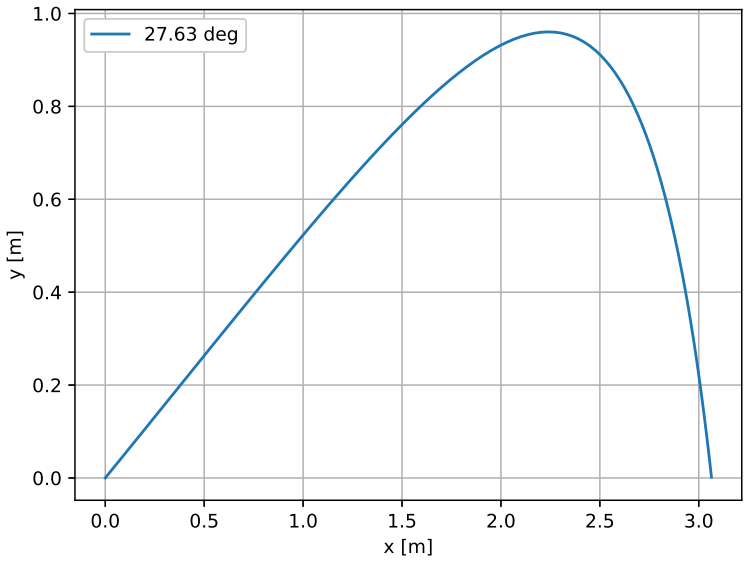
Παρακάτω ακολουθεί ο κώδικας που γράφτηκε σε Python και έγινε χρήση της βιβλιοθήκης Numpy. Η υλοποίηση του αλγόριθμου Runge-Kutta 4ης τάξης δίνεται στο Παράρτημα.

ex3.py

```
1 import numpy as np
  from scipy.optimize import minimize_scalar
  import matplotlib.pyplot as plt
  import diffEquation as de
6
```



Σχήμα 1: Συνάρτηση $R(\phi)$



Σχήμα 2: Τροχιά βλήματος για $\phi = \phi_{max}$

```
def f(t, y):
    g = 10
    c = g/4
    A = np.array([[0, 1, 0, 0],
                  [0, -c, 0, 0],
                  [0, 0, 0, 1],
                  [0, 0, -c, 0]])
    b = np.array([0, 0, 0, -g])
    return np.dot(A, y) + b


def RangeFun(phi, V0, h, f, solver):
    Vx = V0 * np.cos(phi)
    Vy = V0 * np.sin(phi)
    Y0 = np.array([0., Vx, 0., Vy])
    solution = de.solve(Y0, h, f, solver)
    x = solution['x']
    return x[x.size - 1]


plt.close('all')
V0 = 10.  # Initial velocity
h = 0.02

# Find maximum using Brent's Algorithm
g = minimize_scalar(lambda phi: -RangeFun(phi,
                                          V0,
                                          f,
                                          de.rungeKutta4,
                                          h),
                    bracket=(0, np.pi/2))
print("Max Range @%.2f deg: %.4f" % (np.rad2deg(g['x']),
                                     -g['fun']))
plt.figure(1)
phi_ = np.arange(0.,
                 np.pi/2,
                 0.01)  # range of angles to find range
Ranges = list()
i = 0
for phi in phi_:
```

```
    Ranges.insert(i, RangeFun(phi,
                              V0,
                              f,
                              de.rungeKutta4,
                              h))
    i = i + 1
plt.plot(np.rad2deg(phi_),
         np.array(Ranges), '.')  # Plot range vs angle
plt.ylabel('Range [m]')
plt.xlabel('Angle [deg]')
plt.grid()

# Exhaustive Search - Assumes range functions has one maximum
Range = 0.
Ranges = list()
Y = list()
for i, phi in enumerate(phi_):
    Vx = V0 * np.cos(phi)
    Vy = V0 * np.sin(phi)
    Y0 = np.array([0., Vx, 0., Vy])
    solution = de.solve(Y0,
                        f,
                        de.rungeKutta4,
                        h)
    Y.insert(i, solution)
    x = solution['x']
    y = solution['y']
    Ranges.insert(i, x[x.size - 1])

maxRange = max(Ranges)  # Find max element of calculated values
i = Ranges.index(maxRange)  # Find phi corresponding to max
    element
maxPhi = np.rad2deg(phi_[i])

print("Max Range @%.2f deg: %.4f" % (maxPhi,
                                     maxRange))
plt.figure(2)
plt.plot(Y[i]['x'],
         Y[i]['y'],
         label="%.2f deg" % maxPhi)
```

```
86 plt.ylabel('y [m]')
   plt.xlabel('x [m]')
   plt.grid()
   plt.legend()
```