LAPORAN TUGAS BESAR

ALGORITMA DAN PEMROGRAMAN

GERAK PELURU

Laporan ini disusun untuk memenuhi syarat mata kuliah Algoritma dan Pemrograman

Program Studi S1 Teknik Fisika Universitas Telkom

Disusun oleh :

KELOMPOK 9 KELAS TF-47-03

Anggota :

1. Gizza Gratia Lakeisha Gustaman (1010423300049)
2. Levya Johany Windayu (101042330016)
3. Sheila Angela Becalel (101042330077)
4. Yosafat Deguis Hutabarat (101042330078)

FAKULTAS TEKNIK ELEKTRO

TELKOM UNIVERSITY

BANDUNG

2023

DAFTAR ISI

BAB I

PENDAHULUAN

1.1 Latar Belakang

Algoritma adalah tiang kokoh dalam pemograman. Algoritma digunakan untuk menemukan cara menyelesaikan masalah tugas tertentu. Khususnya digunakan untuk pemrosesan data, pengembangan perangkat lunak, analisis data, pemodelan, simulasi dll. Algoritma membantu dalam Menyusun rencana yang terstruktur untuk menyelesaikan masalah secara efisien dan efektif.

Disini kami sebagai penulis membuat program untuk menyelesaikan permasalahan terkait konsep gerak peluru, Pada dasarnya gerak peluru sama dengan gerak parabola. Gerak peluru sendiri adalah suatu jenis gerakan awalnya benda diberi kecepatan awal lalu gerak benda dipengaruhi gaya gravitasi. Gerak parabola adalah gabungan gerak pada arah horizontal dan vertical.

1.2 Tujuan

1. Membuat Program untuk mencari tahu pengaruh kecepatan awal terhadap Gerak vertical keatas dan gerak vertical ke bawah
2. Mengetahui bentuk grafik dari gerak jatuh bebas

BAB II

METODE

2.1 Metode

Lintasan suatu proyektil dalam medan gravitasi yang seragam merupakan masalah fisika klasik yang penting. Namun, kebanyakan kursus fisika mengasusmsikan bahwa gaya gesek dapat diabaikan. Hal ini tentu saja tidak selalu terjadi, terutama pada kecepatan tinggi. Jika gaya gesek diperhitungkan, lintasan proyektil tidak lagi dapat dijelaskan secara analitis, dan lintasannya harus dihitung secara numerik. Untuk melakukan hal ini, seseorang harus menyelesaikan serangkaian persamaan diferensial yang terkait, yang memberikan kesempatan bagus bagi kita untuk menerapkan penyelesaian ODE dalam bentuk vektor. Hal ini ternyata menjadi implementasi yang elegan dan mudah, yang dapat digunakan untuk menyelesaikan persamaan diferensial tingkat tinggi maupun persamaan diferensial yang terkait.

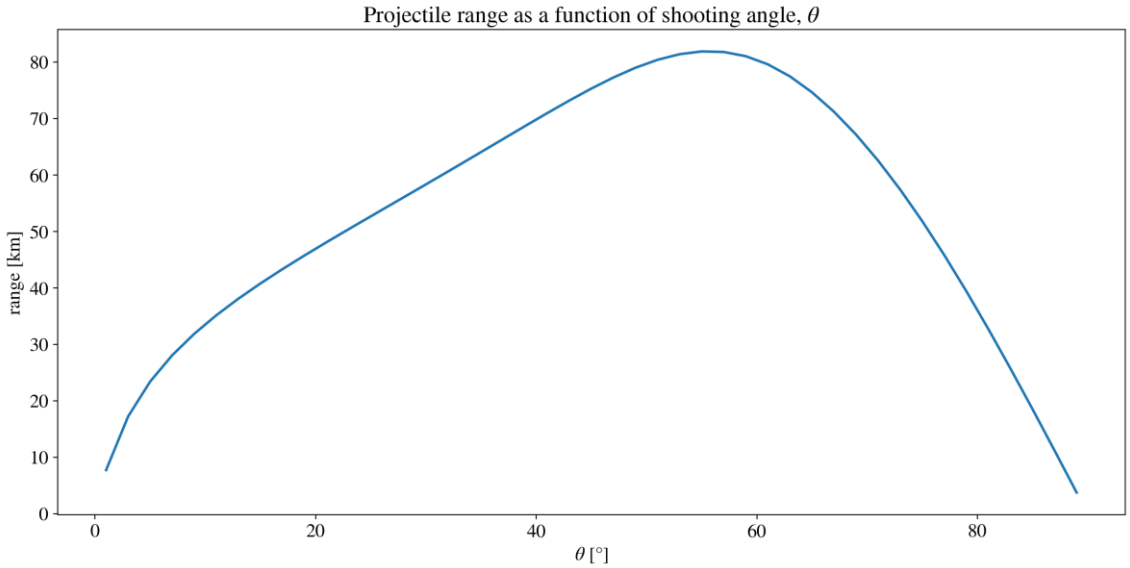
Kode yang kami kami kerjakan ini melakukan simulasi pergerakan proyektil dengan mempertimbangkan berbagai parameter fisika seperti kecepatan awal, kepadatan udara, dan gaya gravitasi. Kode ini menggunakan metode numerik, khususnya metode Runge-Kutta, untuk memecahkan persamaan gerak proyektil. Tujuan utamanya adalah menemukan sudut optimal untuk jangkauan maksimum dan memvisualisasikan lintasan proyektil untuk berbagai sudut tembakan awal.

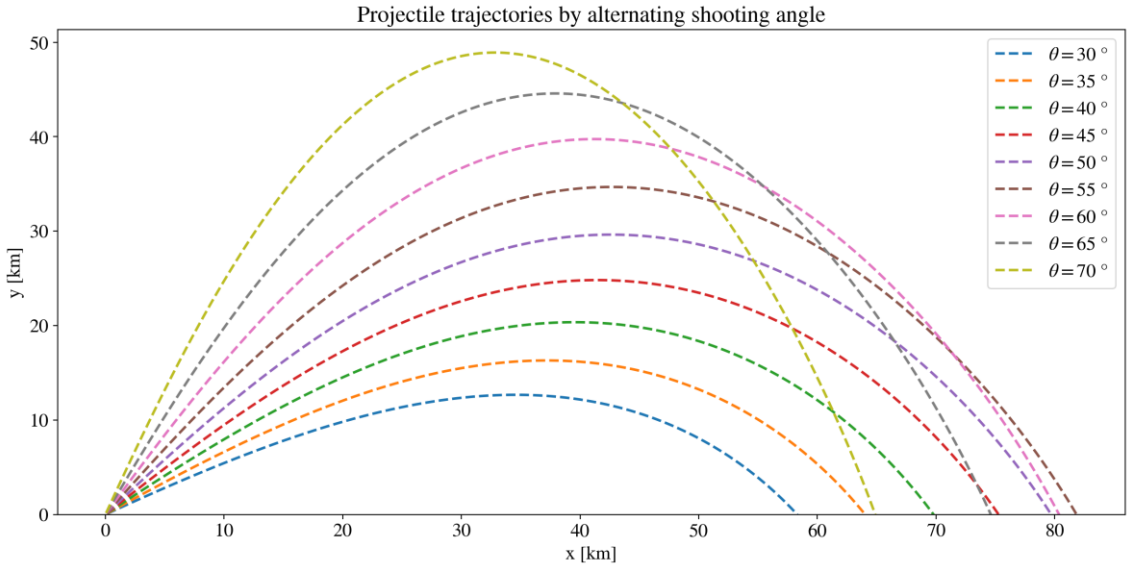
Metode Runge-Kutta orde 4 merupakan metode numerik yang sering digunakan untuk menyelesaikan persamaan diferensial. Berdasarkan ekspansi fungsi dari Deret Taylor, Metode Runge-Kutta orde 4 memiliki galat pemotongan yang minimum sehingga menghasilkan solusi yang baik. Pertama mendefinisikan satu set N nilai waktu diskrit $t_n = t_0 + n \cdot h$ dengan n = 0,1,2,3,..,N. h adalah panjang langkah waktu dan dihitung dengan rumus $h=\frac{t_N-t_0}{N}$, dimana t_0 dan t_n menunjukkan nilai waktu awal dan akhir dari penerbangan proyektil. Ini N nilai waktu sesuai dengan N jumlah x, y, u, dan w nilai, masing-masing dilambangkan x_n, y_n, u_n, dan w_n. Cara yang lebih praktis untuk menunjukkan kumpulan nilai ini adalah pada bentuk vector. tanpa ketergantungan waktu, sehingga $\overrightarrow{w_n} = f(\overrightarrow{w_n})$.

BAB III

HASIL DAN PEMBAHASAN

3.1 Hasil





3.2 Pembahasan

Program ini melakukan simulasi gerak proyektil dengan memecahkan persamaan diferensial yang menggambarkan pergerakan proyektil menggunakan metode Runge-Kutta orde keempat. Pertama, itu menghitung lintasan proyektil untuk sudut tembakan yang berbeda dan menemukan sudut optimal yang memberikan jangkauan maksimum dalam arah horizontal. Kemudian, program ini memplot grafik jarak tempuh proyektil sebagai fungsi dari sudut tembakannya dan juga menggambarkan beberapa lintasan proyektil dengan sudut yang berbeda secara visual.

Kode ini memanfaatkan konsep fisika seperti model kepadatan udara adiabatik, gaya gravitasi, kecepatan awal, dan parameter lainnya untuk memodelkan gerak proyektil dalam keadaan tanpa angin dan menghitung jarak yang dapat dicapai dengan sudut tembakan yang berbeda.

BAB IV

KESIMPULAN

4.1 Kesimpulan

Program ini menggunakan metode Runge-Kutta orde keempat untuk mensimulasikan gerak peluru dengan kondisi tanpa angin. Program ini menghitung, menganalisis pergeraka dari berbagai sudut juga menemukan sudut penghasil jangkauan maksimum dalam arah horizontal. Hasil dari simulasi tersebut lalu divisualkan dalam bentuk grafikjarak tempuh peluru terhadap sudut tembakan dan lintasan peluru lainnya dengan sudut yang berbeda. Dengan memanfaatkan konsep fisika, program ini memberikan visualisasi tentang bagaiman sudut memengaruhi jarak dari sudut tembakannya. Hal ini dapat membantu untuk menvisualisasikan dan memprediksi lintasan dalam kondisi tertentu.

4.2 Kode Program

```
# Importing necessary packages
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
%matplotlib inline

newparams = {'figure.figsize': (15, 7), 'axes.grid': False,
             'lines.markersize': 10, 'lines.linewidth': 2,
             'font.size': 15, 'mathtext.fontset': 'stix',
             'font.family': 'STIXGeneral', 'figure.dpi': 200}
plt.rcParams.update(newparams)

# Constants

g = 9.81              # Force of gravity per kilo on earth's surface
alpha = 2.5           # Parameter in the adiabatic air density model
a = 6.5 * 10 ** (-3) # Parameter in the adiabatic air density model
T_0 = 288             # Temperature at sea level
m = 50                # Mass of the projectile
B = 2 * 10 ** (-3)    # Constant based on the Paris gun
v_0 = 1640            # Initial velocity
```

```
# Wind strength. This is a constant that never varies
V = np.zeros(2) # Starting of with no wind


def f(w):
    """ A function describing the right hand side of the equations
    of motion/the set of ODEs.
    Parameter:
        w       vector containg the needed coordinates and their
velocities
    """
    # V : the vector describing the wind strength and direction
    temp_vector = np.zeros(4)
    temp_vector[0] = w[2]
    temp_vector[1] = w[3]
    # Saving the next parameters in order to optimize run time
    k = (1 - a * w[1] / T_0)   # Air density factor
    s = np.sqrt((w[2]-V[0]) ** 2 + (w[3]-V[1]) ** 2 )
    if k>0:
        temp_vector[2] = - B / m * k ** (alpha) * (w[2]-V[0]) * s
        temp_vector[3] = - B / m * k ** (alpha) * (w[3]-V[1]) * s - g
    else:
        temp_vector[2] = 0
        temp_vector[3] = - g
    return temp_vector


def RK4_step(f, w, h):
    """Performing a single step of the Runge-Kutta fourth order method.
    Parameters:
        f      RHS of ODEs to be integrated
        w      numerical approximation of w at time t
        h      unit/integration step length
    Returns:
        numerical approximation of w at time t+h
    """
    s1 = f(w)
    s2 = f(w + (h / 2) * s1)
    s3 = f(w + (h / 2) * s2)
    s4 = f(w + h * s3)
    return w + (h / 6) * (s1 + (2 * s2) + (2 * s3) + s4)

def shoot(theta, v0):
    """ Initializes the vector w (x and y position and velocity of the
    projectile) given a initial shooting angle, theta, and
    absolute velocity, v0.
    """
```

```
    w = np.zeros(4)
    w[2] = v0 * np.cos(np.deg2rad(theta))
    w[3] = v0 * np.sin(np.deg2rad(theta))
    return w


def projectile_motion(h, theta):
    """ Calculates the motion of the projectile using the functions
    defined above. While the projectile is in the air (w[1] >=0) the
    position and velocity is updated using a single step of RK4.
    Parameters:
        h        unit/integration step length
        theta    initial shooting angle
    Returns:
        X_list   array of the projectile's x-position
        Y_list   array of the porjectile's y-position
    """
    w = shoot(theta, v_0)
    X_list = np.zeros(0)
    Y_list = np.zeros(0)
    while  w[1] >= 0:
        w = RK4_step(f, w, h)
        X_list = np.append(X_list, w[0])
        Y_list = np.append(Y_list, w[1])
    return X_list, Y_list

def find_optimal_angle(h):
    """ Given an integration time step, this function calculates the
optimal initial
    shooting angle for the projectile to obtain maximum range, in x-
direction. The
    set of angles tested, with their corresponding range, along with the
optimal
    angle are returned.
    """


    record = 0        # Placeholder variable that holds the maximum range
    optimal_angle = 0 # Placeholder variable that holds the angle
yielding the maximum range
    # Lists containing the initial angle and its corresponding range
    theta_list = np.zeros(0)
    range_list = np.zeros(0)
    for theta in range (1,90,2):
        x_list, y_list = projectile_motion(h, theta)
        # Using linear interpolation do determine the landing point more
precisely
```

```
        m = (y_list[-1] - y_list[-2]) / (x_list[-1] - x_list[-2])   # The
landing point
        x_range = - y_list[-1] / m + x_list[-1]
        theta_list = np.append(theta_list, theta)
        range_list = np.append(range_list, x_range)
        # Update records
        if x_range >= record:
            record = x_range
            optimal_angle = theta

    # Rerunning the same code on a smaller interval in order to
approximate the optimal angle
    # more precicely
    theta_list_smaller = np.linspace(optimal_angle - 2, optimal_angle +
2, 41)
    for theta_small in theta_list_smaller:
        x_list, y_list = projectile_motion(h, theta)
        # Again, using linear interpolation do determine the landing
point more precisely
        m = (y_list[-1] - y_list[-2]) / (x_list[-1] - x_list[-2])
        x_range = - y_list[-1] / m + x_list[-1]
        if x_range >= record:
            record = x_range
            optimal_angle = theta_small

    return theta_list, range_list, optimal_angle

theta, x , best = find_optimal_angle(0.1)

print("The optimal angle is: ", best, " degrees")

plt.plot(theta, x/1000)
plt.title(r"Projectile range as a function of shooting angle, $\theta$")
plt.xlabel(r"$\theta $ [$\degree$]")
plt.ylabel(r"range [km]")

def trajectories(h):
    plt.figure()
    plt.title("Projectile trajectories by alternating shooting angle")
    plt.xlabel(r"x [km]")
    plt.ylabel(r"y [km]")
    theta_list = np.arange(30.0,75,5)
    for angle in theta_list:
        x_list, y_list = projectile_motion(h, angle)
        plt.plot(x_list/1000, y_list/1000, '--', label=r'$\theta = $%.i
$\degree$'%(angle))
    plt.legend(loc='best')
```

```
    plt.gca().set_ylim(bottom=0)
    plt.show()
trajectories(0.1)
```