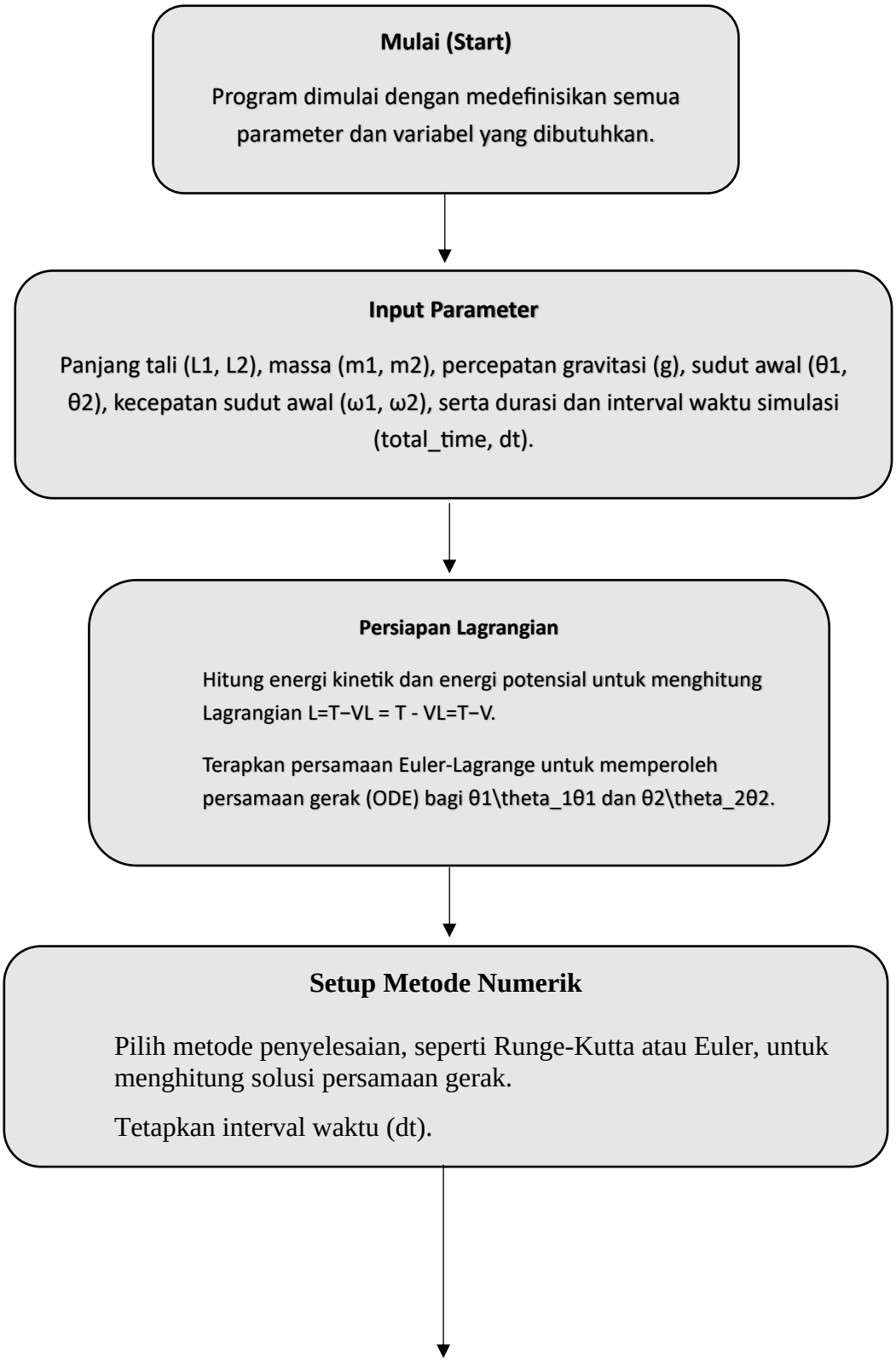
Attala Muflih Gumilang

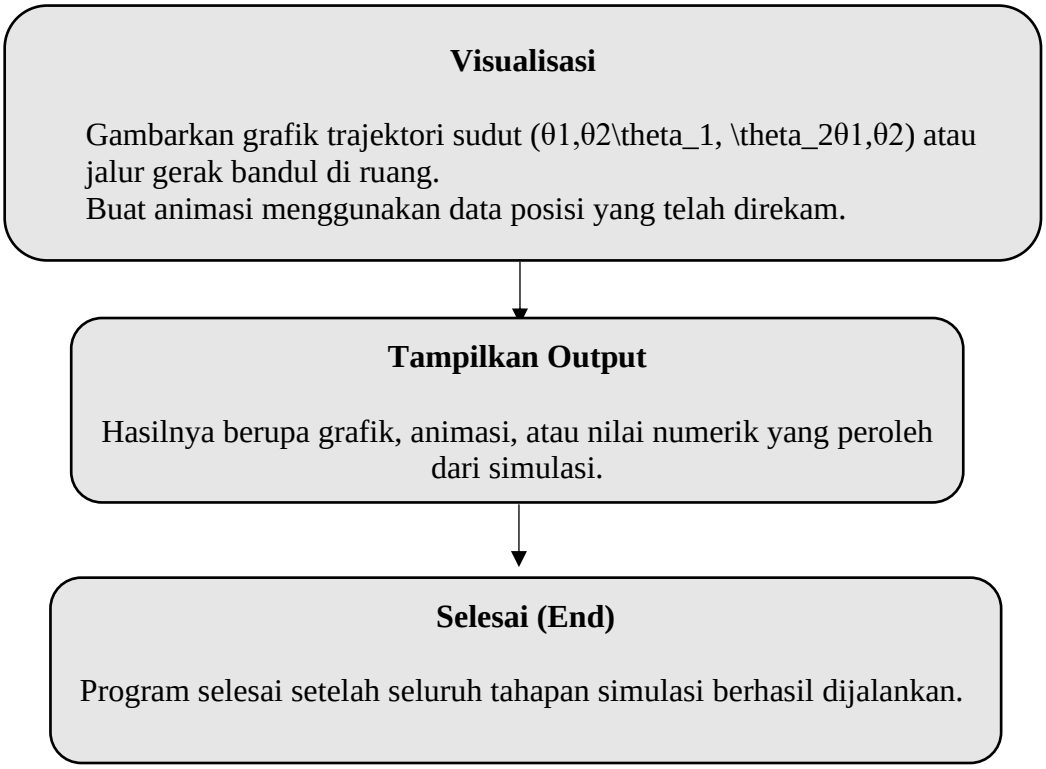
1227030007

Materi-8 Analisis Double Pendulum

Praktikum Fisika Komputasi

Buatkan diagram alir (flowchart) dari kode program double pendulum





Menjelaskan algoritma pada kode program

Struktur Algoritma:

1. Konsep Sistem Pendulum Ganda

Pendulum ganda terdiri dari dua pendulum yang dihubungkan satu sama lain. Dalam kasus ini, kedua pendulum tersebut dipasang pada sebuah kerangka yang bergerak secara harmonik dari sisi ke sisi. Sistem ini sangat kompleks dan menarik karena menunjukkan perilaku non-linear yang dapat menghasilkan gerakan yang sangat tidak teratur (chaotic).

2. Penerapan Metode Lagrange untuk Analisis Sistem

Untuk menganalisis dinamika sistem, digunakan persamaan Lagrange, yang merupakan metode untuk memperoleh persamaan gerak dari suatu sistem fisik dengan mempertimbangkan energi kinetik dan energi potensial.

- Energi Kinetik dihitung berdasarkan kecepatan dari kedua bob pendulum.
- Energi Potensial dihitung berdasarkan ketinggian kedua bob terhadap tanah.
- Lagrangian (L) adalah selisih antara energi kinetik dan energi potensial (L = T - V).
- Dengan menggunakan persamaan Lagrange-Euler, kita mendapatkan persamaan diferensial yang menggambarkan dinamika sistem ini.

3. Menyusun Persamaan Diferensial Gerak

Setelah memperoleh Lagrangian, digunakan metode Lagrange untuk membentuk dua persamaan diferensial terkait sudut (theta1 dan theta2) dari masing-masing pendulum.

- Persamaan ini memperhitungkan efek dari gaya gravitasi, panjang lengan pendulum, massa bob, serta osilasi dasar yang bergerak.

- Solusi dari persamaan ini memberikan percepatan sudut (theta double dot) untuk masing-masing pendulum.

4. Menyelesaikan Persamaan dengan Pendekatan Numerik

Karena persamaan gerak yang dihasilkan terlalu kompleks untuk diselesaikan secara analitik, digunakan metode numerik (odeint dari SciPy) untuk menyelesaikan persamaan diferensial ini.

- Solusi numerik ini memberikan nilai sudut dan kecepatan sudut dari kedua pendulum pada setiap titik waktu yang diinginkan.
- Proses ini dilakukan pada rentang waktu tertentu untuk menghasilkan data posisi dan kecepatan sistem secara kontinu.

5. Menghitung Energi Sistem

Untuk mengevaluasi stabilitas sistem, dihitung energi total (kinetik + potensial) dari sistem. Energi ini digunakan untuk melihat bagaimana variasi frekuensi dasar (w) mempengaruhi sistem.

- Dengan mengubah nilai frekuensi dasar (w), dianalisis bagaimana energi rata-rata sistem berubah.
- Hal ini memberikan wawasan mengenai kondisi resonansi, di mana sistem dapat menyerap energi secara lebih efisien pada frekuensi tertentu.

6. Visualisasi dan Animasi Sistem

Setelah mendapatkan solusi numerik, hasilnya divisualisasikan dalam bentuk animasi menggunakan Matplotlib.

- Animasi menunjukkan bagaimana posisi kedua bob pendulum berubah seiring waktu.
- Kerangka yang bergerak divisualisasikan sebagai titik yang berosilasi dari sisi ke sisi, sementara kedua bob pendulum digambarkan dengan garis yang menghubungkan kerangka tersebut.
- Juga ditambahkan jejak (trail) untuk menunjukkan lintasan yang dilalui oleh bob pendulum, memberikan efek visual yang memperlihatkan dinamika kompleks sistem ini.

7. Eksplorasi Resonansi dan Bifurkasi

Simulasi ini juga digunakan untuk mengeksplorasi fenomena resonansi dan bifurkasi.

- Ketika frekuensi dasar (w) mendekati nilai tertentu, pendulum dapat memasuki kondisi resonansi, yang menyebabkan osilasi menjadi lebih besar.
- Pada beberapa frekuensi, sistem dapat menunjukkan perilaku chaos, di mana gerakannya menjadi sangat tidak teratur dan sensitif terhadap kondisi awal.

Analisis grafik dan animasi double pendulum

- Grafik Sudut terhadap Waktu (θ1, θ2):

Grafik ini menggambarkan bagaimana sudut bandul pertama (θ1) dan kedua (θ2) berubah seiring berjalannya waktu. Awalnya, sudut-sudut ini bergerak secara teratur, namun dengan cepat beralih ke pola kacau karena sifat chaos dan sensitivitas terhadap kondisi awal.

- Grafik Energi Sistem:

Energi total dari sistem, dalam kondisi ideal, tetap konstan, mencerminkan keseimbangan energi. Fluktuasi energi kinetik dan potensial menunjukkan adanya transfer energi antara kedua bandul.

- Diagram Fase:

Grafik yang menampilkan hubungan antara sudut (θ) dan kecepatan sudut (ω) memperlihatkan pola fase yang rumit dan tak beraturan, mencerminkan karakteristik sistem yang chaotic.

- Animasi Pendulum Ganda:

Animasi ini memperlihatkan peralihan gerakan dari yang awalnya teratur menjadi kacau, sekaligus menunjukkan dinamika sistem dan bagaimana energi didistribusikan secara visual selama simulasi.

- Pola Chaos:

Pendulum ganda menunjukkan kepekaan ekstrem terhadap perubahan kecil dalam kondisi awal, menghasilkan pola gerak yang sangat tidak stabil dan sulit diprediksi.

Lampiran Kode Pemrograman

```
import numpy as np

import sympy as smp

from scipy.integrate import odeint

import matplotlib.pyplot as plt

from matplotlib import animation

from mpl_toolkits.mplot3d import Axes3D

from matplotlib.animation import PillowWriter

from IPython.display import HTML


t, m, g, L1, L2, w, C, alph, beta = smp.symbols(r't m g L_1, L_2 \omega C \alpha \beta')


the1, the2, = smp.symbols(r'\theta_1 \theta_2 ', cls=smp.Function)


the1 = the1(t)

the1_d = smp.diff(the1, t)

the1_dd = smp.diff(the1_d, t)


the2 = the2(t)

the2_d = smp.diff(the2, t)

the2_dd = smp.diff(smp.diff(the2, t), t)


x1, y1, x2, y2 = smp.symbols('x_1, y_1, x_2, y_2', cls=smp.Function)

x1 = x1(t, the1)
```

```
y1 = y1(t, the1)

x2 = x2(t, the1, the2)

y2 = y2(t, the1, the2)



x1 = smp.cos(w*t) + L1 * smp.sin(the1)

y1 = -L1 * smp.cos(the1)

x2 = smp.cos(w*t) + L1 * smp.sin(the1) + L2 * smp.sin(the2)

y2 = -L1 * smp.cos(the1) - L2 * smp.cos(the2)



smp.diff(x1, t)



vx1_f = smp.lambdify((t, w, L1, L2, the1, the2, the1_d, the2_d), smp.diff(x1, t))

vx2_f = smp.lambdify((t, w, L1, L2, the1, the2, the1_d, the2_d), smp.diff(x2, t))

vy1_f = smp.lambdify((t, w, L1, L2, the1, the2, the1_d, the2_d), smp.diff(y1, t))

vy2_f = smp.lambdify((t, w, L1, L2, the1, the2, the1_d, the2_d), smp.diff(y2, t))




T = 1/2 * (smp.diff(x1, t)**2 + smp.diff(y1, t)**2) + \

    1/2 * m * (smp.diff(x2, t)**2 + + smp.diff(y2, t)**2)

V = g * y1 + m * g * y2

L = T - V



# Persamaan Lagrange-Euler untuk theta1

LE1 = smp.diff(L, the1) - smp.diff(smp.diff(L, the1_d), t)
```

```
LE1 = LE1.simplify()



# Persamaan Lagrange-Euler untuk theta2

LE2 = smp.diff(L, the2) - smp.diff(smp.diff(L, the2_d), t)

LE2 = LE2.simplify()



sols = smp.solve([LE1, LE2], (the1_dd, the2_dd),

            simplify=False, rational=False)



sols[the1_dd]



a = LE1.subs([(smp.sin(the1 - the2), the1 - the2),

   (smp.cos(the1 - the2), 1),

   (smp.cos(the1), 1),

   (smp.sin(the1), the1),

   (the1, C * smp.cos(w * t)),

   (the2, C * alph * smp.cos(w * t)),

   (m, 1),

   (L2, L1),

]).doit().series(C, 0, 2).removeO().simplify()



b = LE2.subs([(smp.sin(the1 - the2), the1 - the2),

   (smp.cos(the1 - the2), 1),

   (smp.cos(the1), 1),
```

```
    (smp.sin(the1), the1),
    (smp.sin(the2), the2),
    (the1, C * smp.cos(w * t)),
    (the2, C * alph * smp.cos(w * t)),
    (m, 1),
    (L2, L1),
]).doit().series(C, 0, 2).removeO().simplify()



yeet = smp.solve([a.args[1], b.args[2]], (w, alph))
yeet [2][0]


yeet[0][0]


smp.limit(yeet[1][0].subs(C, beta/L1).simplify(), beta, smp.oo)


dz1dt_f = smp.lambdify((t, m, g, w, L1, L2, the1, the2, the1_d, the2_d), sols[the1_dd])
dthe1dt_f = smp.lambdify(the1_d, the1_d)


dz2dt_f = smp.lambdify((t, m, g, w, L1, L2, the1, the2, the1_d, the2_d), sols[the2_dd])
dthe2dt_f = smp.lambdify(the2_d, the2_d)



def dSdt(S, t):
```

```
  the1, z1, the2, z2 = S
  return [
     dthe1dt_f(z1),
     dz1dt_f(t, m, g, w, L1, L2, the1, the2, z1, z2),
     dthe2dt_f(z2),
     dz2dt_f(t, m, g, w, L1, L2, the1, the2, z1, z2),
  ]


t = np.linspace(0, 20, 1000)
g = 9.81
m = 1
L1 = 20
L2 = 20
w = np.sqrt(g/L1)
ans = odeint(dSdt, y0=[0, 0, 0, 0], t=t)




plt.plot(ans.T[0])


def get_energy(w):
    t = np.linspace(0, 100, 2000)
    ans = odeint(dSdt, y0=[0.1, 0.1, 0, 0], t=t)
    vx1 = vx1_f(t,w,L1,L2,ans.T[0],ans.T[2],ans.T[1],ans.T[3])
    vx2 = vx2_f(t,w,L1,L2,ans.T[0],ans.T[2],ans.T[1],ans.T[3])
```

```
    vy1 = vy1_f(t,w,L1,L2,ans.T[0],ans.T[2],ans.T[1],ans.T[3])

    vy2 = vy2_f(t,w,L1,L2,ans.T[0],ans.T[2],ans.T[1],ans.T[3])

    E = 1/2 * np.mean(vx1**2 + vx2**2 + vy1**2 + vy2**2)

    return E


ws = np.linspace(0.4, 1.3, 100)

Es = np.vectorize(get_energy)(ws)


plt.plot(ws, Es)

plt.axvline(1.84775 * np.sqrt(g / L1), c='k', ls='--')

plt.axvline(0.76536 * np.sqrt(g / L1), c='k', ls='--')


plt.grid()



t = np.linspace(0, 200, 20000)

g = 9.81

m = 1

L1 = 20

L2 = 20

w = ws[ws>1][np.argmax(Es[ws>1])]

ans = odeint(dSdt, y0=[0.1, 0.1, 0, 0], t=t)


def get_x0y0x1y1x2y2(t, the1, the2, L1, L2):
```

```
    return (np.cos(w*t),
            0*t,
            np.cos(w*t) + L1*np.sin(the1),
            -L1*np.cos(the1),
            np.cos(w*t) + L1*np.sin(the1) + L2*np.sin(the2),
            -L1*np.cos(the1) - L2*np.cos(the2)
        )

x0, y0, x1, y1, x2, y2 = get_x0y0x1y1x2y2(t, ans.T[0], ans.T[2], L1, L2)

def animate(i):
    ln1.set_data([x0[::10][i], x1[::10][i], x2[::10][i]], [y0[::10][i], y1[::10][i], y2[::10][i]])
    trail1 = 50
    trail2 = 50
    ln2.set_data(x1[::10][i:max(1,i-trail1):-1], y1[::10][i:max(1,i-trail1):-1]) # Use the built-in function max()
    ln3.set_data(x2[::10][i:max(1,i-trail2):-1], y2[::10][i:max(1,i-trail2):-1]) # Use the built-in function max()

fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.set_facecolor('k')

ax.get_xaxis().set_ticks([])
ax.get_yaxis().set_ticks([])
```

```
ln1, = ax.plot([], [], 'ro--', lw=3, markersize=8)

ln2, = ax.plot([], [], 'ro-', markersize=8, alpha=0.05, color='cyan')

ln3, = ax.plot([], [], 'ro-', markersize=8, alpha=0.05, color='cyan')



ax.set_ylim(-44, 44)

ax.set_xlim(-44, 44)




ani = animation.FuncAnimation(fig, animate, frames=2000, interval=50)

HTML(ani.to_html5_video())
```

Link google collab:

https://colab.research.google.com/drive/1PamC7SEOiObof7MYXgLkz7JCK6v-vEV1#scrollTo=VGHDIwFFGrig