Modelowanie i symulacja systemów

Dom 1

Nikita Florek

Florek

Wahadło.

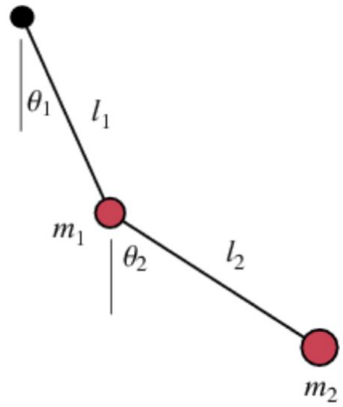
Wahadło – ciało zawieszone w jednorodnym polu grawitacyjnym w taki sposób, że może wykonywać drgania wokół poziomej osi nie przechodzącej przez środek ciężkości zawieszonego ciała.

W mechanice rozróżnia się dwa podstawowe rodzaje wahadeł:

- matematyczne (proste),
- fizyczne.

Ważną cechą wahadeł fizycznego i matematycznego jest niemal pełna niezależność ich okresu drgań od amplitudy, co jest dobrze spełnione dla małych wychyle. Własność ta, zwana izochronizmem drgań, została odkryta około 1602 roku przez Galileusza, który używał wahadła do pomiaru czasu. Zainspirowany tą zasadą Christiaan Huygens zbudował w 1656 roku pierwszy zegar wahadłowy. Zegary wahadłowe były najdokładniejszymi urządzaniami do pomiaru czasu aż do skonstruowania w latach 30. XX wieku zegarów kwarcowych. W ogólności wahadło jest oscylatorem anharmonicznym, jego okres drgań i inne parametry zależy od amplitudy. Opis matematyczny rozwiązań równania ruchu wahadła jest w ogólności dość złożony, ale założenia upraszczające przyjmowane dla małych amplitud drgań pozwalają rozwiązać równania ruchu w sposób analityczny.

Wahadło podwójne.



Równania ruchu wahadła możemy otrzymać z równań Eulera-Lagrange'a. Lagranżjan $L = T - V$ (Zadanie).

$$(m_1 + m_2)l_1\ddot{\theta}_1 + m2l_2\ddot{\theta}_2\cos(\delta) + m_2l_2\dot{\theta}_2^2\sin(\delta) + g(m_1 + m_2)\sin\theta_1 = 0\,,$$
$$m_2l_2\ddot{\theta}_1 + m_2l_1\ddot{\theta}_1\cos(\delta) - m_2l_1\dot{\theta}_1^2\sin(\delta) + m_2g\sin\theta_2 = 0\,,$$

gdzie $\delta = \theta_1 - \theta_2$.

Stąd

$$\begin{aligned}\ddot{\theta}_1 &= -[gm\sin\theta_1 - gm_2\cos(\delta)\sin\theta_2 + l_1m_2\cos(\delta)\sin(\delta)\dot{\theta}_1^2] + l_2m_2\sin(\delta)\dot{\theta}_2^2]/D_1\\ \ddot{\theta}_2 &= [gm\cos(\delta)\sin\theta_1 - gm\sin\theta_2 + l_1m\sin(\delta)\dot{\theta}_1^2 + l_2m_2\cos(\delta)\sin(\delta)\dot{\theta}_2^2]/D_2\end{aligned}$$

gdzie $D_1 = l_1(m - m_2\cos^2\delta)$, $D_2 = l_2/l_1D_1$.

Kładąc $\theta_1 = y_0, \dot{\theta}_1 = y_1$, $\theta_2 = y_3, \dot{\theta}_2 = y_4$, równania te zapiszemy w postaci

$$\begin{aligned}\dot{y}_0 &= y_1\\ \dot{y}_1 &= -[gm\sin y_0 - gm_2\cos(\delta)\sin y_2 + l_1m_2\cos(\delta)\sin(\delta)y_1^2] + l_2m_2\sin(\delta)y_4]/D_1\\ \dot{y}_2 &= y_3\\ \dot{y}_3 &= [gm\cos(\delta)\sin y_0 - gm\sin y_3 + l_1m\sin(\delta)y_1^2 + l_2m_2\cos(\delta)\sin(\delta)y_4^2]/D_2\end{aligned}$$

gdzie $\delta = y_0 - y_3$, a D_1 i D_2 są dane jak poprzednio. Postać ta jest odpowiednia dla całkowania numerycznego.

Używamy dowolnej metody Rungego-Kutty; 4-go lub wyższego rzędu. Jeden krok całkowania równania 1-go rzędu

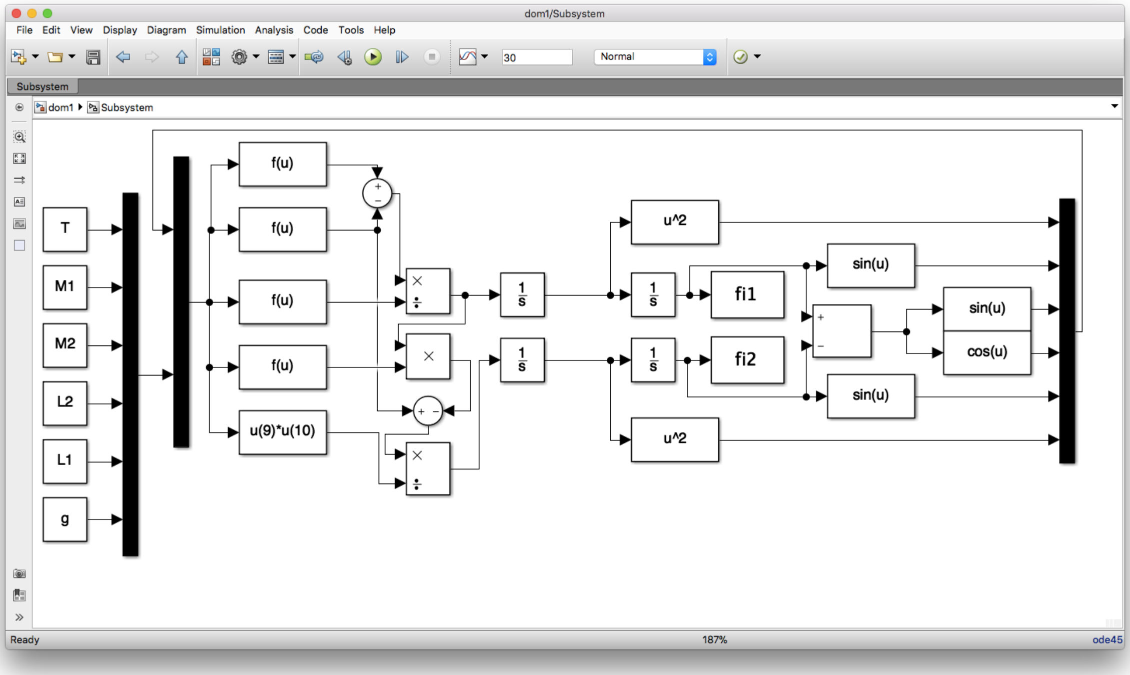
$$\dot{y} = f(t, y)\,, \qquad y(t_0) = y_0.$$

z krokiem h, metodą, np. RK4:

$$\begin{aligned}
k_1 &= hf(t, y_t) \\
k_2 &= hf(t + h/2, y_t + k_1/2) \\
k_3 &= hf(t + h/2, y_t + k_2/2) \\
k_4 &= hf(t + h, y_t + k_3) \\
y_{t+h} &= y_t + (k_1 + 2k_2 + 2k_3 + k_4)/6
\end{aligned}$$

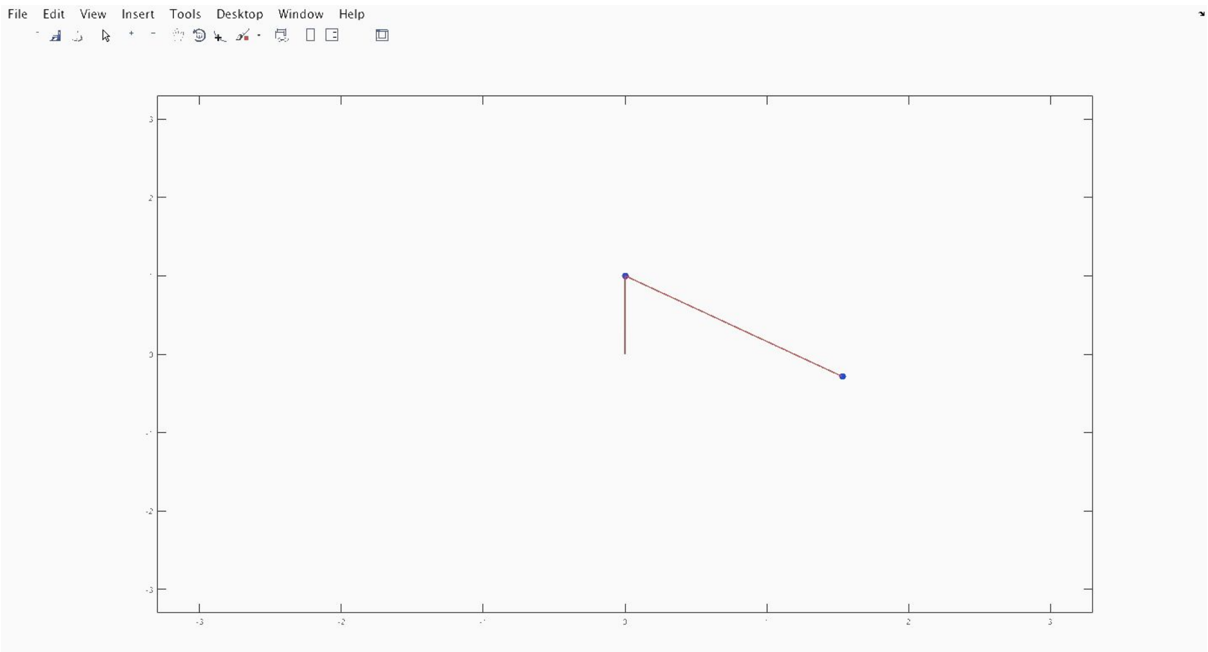
Tutaj y_t jest numeryczną wartością $y(t)$ (przybliżoną).

Simulink.


dom1/Subsystem
File Edit View Display Diagram Simulation Analysis Code Tools Help
30
Normal
Subsystem
dom1
Subsystem
T
M1
M2
L2
L1
g
f(u)
f(u)
f(u)
f(u)
u(9)*u(10)
1
s
1
s
u^2
1
s
1
s
fi1
fi2
sin(u)
sin(u)
cos(u)
sin(u)
u^2
Ready
187%
ode45

Matlab.

```
clear all;
clc;

% uruchomienie symulacji w Simulink'u
sim('dom1')

% Pobranie stałych wartości z Simulink'a
M1 = str2num(get_param('dom1/Subsystem', 'M1'));
M2 = str2num(get_param('dom1/Subsystem', 'M2'));
L1 = str2num(get_param('dom1/Subsystem', 'L1'));
L2 = str2num(get_param('dom1/Subsystem', 'L2'));

% Transponowanie macierzy
fi1=fi1';
fi2=fi2';

% Rysowanie
set(gcf, 'Position', get(0, 'Screensize'));

for i=1:length(fi1)
    hold off
    plot(L1*sin(fi1(i)),-L1*cos(fi1(i)), 'b', 'MarkerSize',M1*30,'Marker','.','LineWidth',2);
    hold on
    plot(L1*sin(fi1(i))+L2*sin(fi2(i)),-L1*cos(fi1(i))-L2*cos(fi2(i)), 'b', 'MarkerSize',M1*30,'Marker','.','LineWidth',2);
    hold on
    axis([-1.1*(L1+L2) 1.1*(L1+L2) -1.1*(L1+L2) 1.1*(L1+L2)]);
    line([0,L1*sin(fi1(i)),L1*sin(fi1(i))+L2*sin(fi2(i))], [0,-L1*cos(fi1(i)),-L1*cos(fi1(i))-L2*cos(fi2(i))],'Color','r','LineWidth',2);
    pause(0.05);
end
```



Python.

```
#!/usr/bin/env python

#################################################################
#  Dom
#
#  Copyright 2018 niquit. All rights reserved.
#################################################################

#################################################################
#  BEGIN CONFIGURE SCRIPT

from numpy import sin, cos
import numpy as numpy
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plot
import scipy.integrate as integrate
import matplotlib.animation as animation

#  END CONFIGURE SCRIPT
#################################################################

#################################################################
#  BEGIN DEFAULT VARIABLES

G = 9.81
L1 = 1.0
L2 = 2.0
M1 = 1.0
M2 = 2.0
fi1 = 180.0
fi2 = 50.0

#  END DEFAULT VARIABLES
#################################################################

#################################################################
#  BEGIN RUNGE-KUTTA FOURTH ORDER METHOD

def derivation(state, t):
    dydx = numpy.zeros_like(state)

    dydx[0] = state[1]
    dydx[1] = (M2 * L1 * state[1] * state[1] * sin(state[2] - state[0]) * cos(state[2] - state[0]) + M2 * G * sin(state[2]) * cos(state[2] - state[0])
+ M2 * L2 * state[3] * state[3] * sin(state[2] - state[0]) - (M1 + M2) * G * sin(state[0])) / ((M1 + M2) * L1 - M2 * L1 * cos(state[2] - state[0])
* cos(state[2] - state[0]))
    dydx[2] = state[3]
    dydx[3] = (-M2 * L2 * state[3] * state[3] * sin(state[2] - state[0]) * cos(state[2] - state[0]) + (M1 + M2) * G * sin(state[0]) * cos(state[2] -
state[0]) - (M1 + M2) * L1 * state[1] * state[1] * sin(state[2] - state[0]) - (M1 + M2) * G * sin(state[2])) / ((L2 / L1) * ((M1 + M2) * L1 - M2 *
L1 * cos(state[2] - state[0]) * cos(state[2] - state[0])))

    return dydx

#  END RUNGE-KUTTA FOURTH ORDER METHOD
#################################################################
```

```
################################################################
#  BEGIN SCRIPT

y = integrate.odeint(derivation, numpy.radians([fi1, 0, fi2, 0]), numpy.arange(0.0, 30, 0.01))

x1 = L1 * sin(y[:, 0])
x2 = L2 * sin(y[:, 2]) + L1 * sin(y[:, 0])

y1 = -L1 * cos(y[:, 0])
y2 = -L2 * cos(y[:, 2]) - L1 * cos(y[:, 0])

figure = plot.figure()
ax = figure.add_subplot(111, autoscale_on=False, xlim=(-1.1*(L1+L2), 1.1*(L1+L2)), ylim=(-1.1*(L1+L2), 1.1*(L1+L2)))
line, = ax.plot([], [], 'o-', lw=2)

def init():
  line.set_data([], [])
  return line,

def animate(i):
  line.set_data([0, x1[i], x2[i]], [0, y1[i], y2[i]])
  line.set_color("red")
  return line,

animation = animation.FuncAnimation(figure, animate, 600, interval=30, init_func=init)

fullscreen = plot.get_current_fig_manager()
fullscreen.resize(*fullscreen.window.maxsize())

plot.show()

#  END SCRIPT
################################################################
```