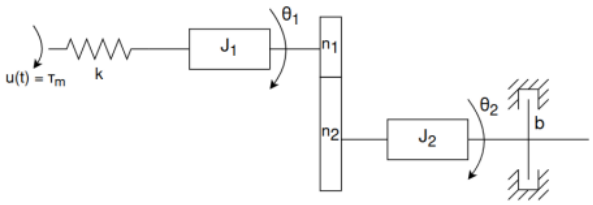
Sprawozdanie z projektu Metody Modelowania Matematycznego

Jakub Sawicki (193197) i Michał Eisler (193258) ACiR3

1 Treść zadania

Projekt 4. Dany jest układ mechaniczny przedstawiony na poniższym rysunku:



Należy wyprowadzić model układu oraz zaimplementować go w symulacji. Symulator powinien umożliwiać pobudzenie układu przynajmniej trzema rodzajami synagłów wejściowych (prostokątny o skończonym czasie trwania, trójkątny, harmoniczny). Symulator powinien umożliwiać zmianę wszystkich parametrów układu oraz sygnałów wejściowych. Należy użyć metody Rungego-Kutty 4-go rzędu oraz metody Eulera oraz na wspólnym wykresie pokazać wyniki symulacji (prędkości i położenia wału J_2) z obu tych metod.

Rysunek 1: Treść zadania

2 Sposób modelowania

Aby zamodelować pokazany na powyższym rysunku układ, trzeba go było opisać równaniem różniczkowy.

$$u(t) - k\theta_1 - J_1\frac{d^2\theta_1}{dt^2} - J_2\frac{d^2\theta_2}{dt^2} - b\frac{d\theta_2}{dt} = 0 \tag{1}$$

Znając liczbę zębów przekładni n1 i n2, można uzależnić od siebie prędkości kątowe dwóch wałów.

$$\frac{\theta_1}{\theta_2} = \frac{n_2}{n_1} \tag{2}$$

$$\theta_1 = \theta_2\frac{n_2}{n_1} \tag{3}$$

Następnie podstawiamy tą zależność do równania różniczkowego.

$$u(t) - k\frac{n_2}{n_1}\theta_2 - (J_1\frac{n_2}{n_1} + J_2)\frac{d^2\theta_2}{dt^2} - b\frac{d\theta_2}{dt} = 0 \tag{4}$$

Aby zasymulować te równanie różniczkowe drugiego rzędu za pomocą wymaganych metod, trzeba je najpierw rozbić na dwa równania różniczkowe pierwszego rzędu.

$$\begin{cases} \omega_2 = \frac{d\theta_2}{dt}, \\ \mathcal{E}_2 = \frac{u(t)}{J_1 \frac{n_2}{n_1} + J_2} - \frac{k}{J_1 + J_2 \frac{n_1}{n_2}} \theta_2 - \frac{b}{J_1 \frac{n_2}{n_1} + J_2} \omega_2 \end{cases} \tag{5}$$

Tak opisany układ jest gotowy do symulacji. Trzeba tylko podstawić wartości parametrów do równania, jak i również podać sygnał pobudzający. Zostały przyjęte zerowe warunki początkowe.

3 Metoda Eulera

Metoda Eulera polega na przybliżeniu rozwiązania równania różniczkowego w dyskretnych punktach, przyjmując mały krok czasowy. W tej metodzie następne rozwiązanie to suma poprzedniego i jego pochodnej. Poniżej przedstawiono implementację metody Eulera dla zadanego przykładu w pythonie.

```
class Eu_simulation(Simulation):
    def eu(self):
        h = self.parameters.step_size
        t = 0
        x = 0

        for i in range(int(self.number_of_steps)):
            self.theta.append(self.theta[i] + h*self.rot_speed[i])
            self.rot_speed.append(self.rot_speed[i] +
                h*(self.f2(t, self.theta[i],
            self.rot_speed[i])))

            t = t + h
            self.time.append(t)

        return self.theta, self.rot_speed, self.time
```

Pętla oblicza kąt na podstawie sumy poprzedniego kąta i pochodnej, czyli prędkości kątowej, oraz prędkość kątową na podstawie poprzedniej prędkości oraz pochodnej, czyli przyspieszenia kątowego (f2()).

4 Metoda Rungego-Kutty czwartego rzędu

Metoda Rungego-Kutty czwartego rzędu polega na liczeniu nachylenia funkcji w przybliżonych krokach k_1, k_2, k_3 i k_4. Każdy krok polega na kroku wcześniejszym, co pozwala na zwiększanie dokładności za każdym krokiem. Na końcu podczas obliczania następnego punktu, bierze się pod uwagę wszystkie wyliczone kroki z pewną wagą. Cały proces wygląda w następujący sposób:

$$\begin{cases} k_1 = f(x_n, y_n) \\ k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} k_1) \\ k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} k_2) \\ k_4 = f(x_n + h, y_n + h k_3) \\ y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{cases} \tag{6}$$

Jako że równanie różniczkowe opisujące symulowany układ jest równaniem drugiego rzędu, to trzeba obliczyć przybliżone kroki k dwa razy, dla równania prędkości kątowej

i dla równania przyśpieszenia kątowego. Każdy kolejny krok korzysta z poprzedniego, więc trzeba je liczyć naprzemiennie, w następujący sposób:

$$\begin{cases} k_{1\theta} = \omega_2(t, \theta_n, \omega_n) \\ k_{1\omega} = \mathcal{E}_2(t, \theta_n, \omega_n) \\ \\ k_{2\theta} = \omega_2(t + \frac{h}{2}, \theta + \frac{h}{2}k_{1\theta}, \omega_n + \frac{h}{2}k_{1\omega}) \\ k_{2\omega} = \mathcal{E}_2(t + \frac{h}{2}, \theta + \frac{h}{2}k_{1\theta}, \omega_n + \frac{h}{2}k_{1\omega}) \\ \\ k_{3\theta} = \omega_2(t + \frac{h}{2}, \theta + \frac{h}{2}k_{2\theta}, \omega_n + \frac{h}{2}k_{2\omega}) \\ k_{3\omega} = \mathcal{E}_2(t + \frac{h}{2}, \theta + \frac{h}{2}k_{2\theta}, \omega_n + \frac{h}{2}k_{2\omega}) \\ \\ k_{4\theta} = \omega_2(t + h, \theta_n + k_{3\theta}, \omega_n + k_{3\omega}) \\ k_{4\omega} = \mathcal{E}_2(t + h, \theta_n + k_{3\theta}, \omega_n + k_{3\omega}) \\ \\ \theta_{n+1} = \theta_n + \frac{h}{6}(k_{1\theta} + 2k_{2\theta} + 2k_{3\theta} + k_{4\theta}) \\ \omega_{n+1} = \omega_n + \frac{h}{6}(k_{1\omega} + 2k_{2\omega} + 2k_{3\omega} + k_{4\omega}) \end{cases} \tag{7}$$

Poniżej przedstawiono implementację metody Rungego-Kutty czwartego rzędu w pythonie.

```
class RKSimulation(Simulation):
    def rk4(self):
        h = self.parameters.step_size
        x = 0
        v = 0
        t = 0
        for i in range(int(self.number_of_steps)):
            K1x = self.f1(t, x, v)
            K1v = self.f2(t, x, v)
            K2x = self.f1(t + h/2, x + h*K1x/2, v + h*K1v/2)
            K2v = self.f2(t + h/2, x + h*K1x/2, v + h*K1v/2)
            K3x = self.f1(t + h/2, x + h*K2x/2, v + h*K2v/2)
            K3v = self.f2(t + h/2, x + h*K2x/2, v + h*K2v/2)
            K4x = self.f1(t + h, x + h*K3x, v + h*K3v)
            K4v = self.f2(t + h, x + h*K3x, v + h*K3v)
            x = x + h/6*(K1x + 2*K2x + 2*K3x + K4x)
            v = v + h/6*(K1v + 2*K2v + 2*K3v + K4v)
            t = t + h

            self.theta.append(x)
            self.rot_speed.append(v)
            self.time.append(t)

        return self.theta, self.rot_speed, self.time
```

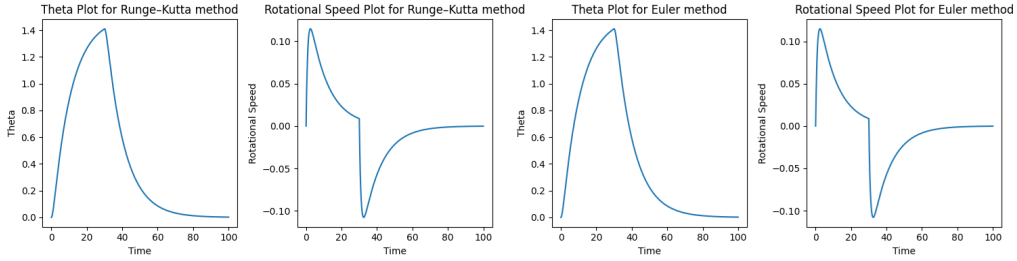
5 Prezentacja danych

Program został napisany w środowisku Visual Studio Code w języku python. Do wykonania programu wykorzystane zostały biblioteki numpy, matplotlib oraz tkinter. Biblioteka numpy pozwoliła łatwiej wykonywać operacje matematyczne. Matplotlib został

wykorzystany do prezentacji wyników - wyświetlania wykresów pokazujących wyniki obu metod. Tkinter posłużył do stworzenia interfejsu graficznego dla programu.

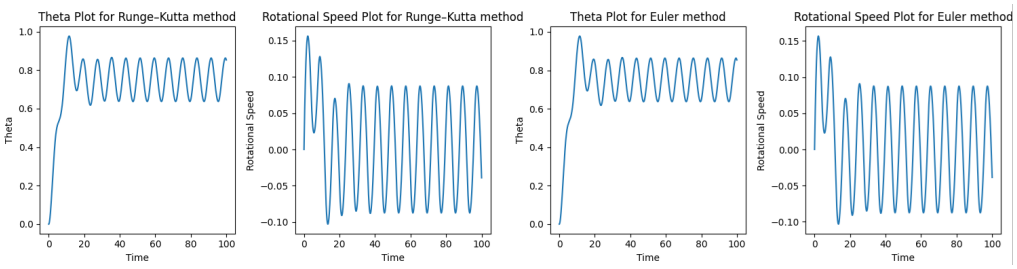
6 Przykładowe wyniki symulacji

Na rysunku drugim został pokazany wykres symulacji położenia i prędkości kątowej wału J2, dla układu pobudzonego sygnałem prostokątnym o czasie trwania 30 i amplitudzie 2.



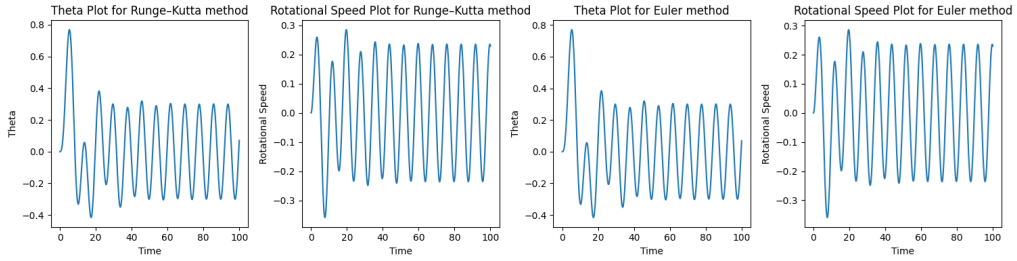
Rysunek 2: Sygnał prostokątny

Na rysunku trzecim pokazana jest symulacja układu pobudzonego sygnałem trójkątnym o okresie 8 i amplitudzie 2.

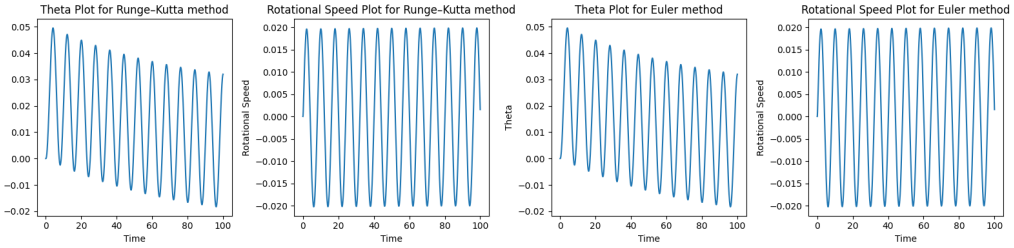


Rysunek 3: Sygnał trójkątny

Na przykładowych sygnałach harmonicznych pokazany jest wpływ parametrów na wyniki symulacji. Na rysunku czwartym widać sygnał harmoniczny z parametrem tłumienia b = 2. Następnie ten współczynnik został zwiększony do 100, co spowodowało zwiększone tłumienie drgać, czyli amplituda drgań prędkości i położenia kątowego znacznie zmalała, co można zaobserwować na rysunku czwartym.



Rysunek 4: Sygnał harmoniczny, b = 2



Rysunek 5: Sygnał harmoniczny, b = 100

7 Wnioski końcowe

Symulacja jest poprawna i czytelna, o ile dobrze się dobierze parametry, takie jak długość kroku czasowego symulacji lub okres.

Program można by poprawić pod kątem interfejsu graficznego, ponieważ nie jest najbardziej estetyczny, lecz spełnia on swoją funkcję i nie wpływa negatywnie na wyniki symulacji.