Raport z projektu w ramach przedmiotu Modelowanie Matematyczne

Bartosz Chabros 198404 Beniamin Cymanowski 198067

2 czerwca 2025

1 Cel projektu

Celem projektu było stworzenie aplikacji umożliwiającej symulację odpowiedzi układu dynamicznego z regulatorem PI na różne typy sygnałów wejściowych. Użytkownik może zmieniać parametry transmitancji, regulatora oraz sygnału wejściowego z poziomu interfejsu graficznego.

2 Opis układu

Układ składa się z transmitancji obiektu $G_p(s)$ oraz regulatora PI $G_c(s)$. Całość została zamknięta w pętli sprzężenia zwrotnego.

2.1 Transmitancja obiektu

$$G_p(s) = \frac{a_1 s + a_0}{b_2 s^2 + b_1 s + b_0}$$

2.2 Regulator PI

$$G_c(s) = K_p + \frac{K_i}{s}$$

3 Interfejs użytkownika

Aplikacja została napisana w języku Python z użyciem bibliotek PyQt6 oraz Matplotlib. Umożliwia ona:

- Wprowadzanie parametrów transmitancji i regulatora,
- Wybór typu sygnału wejściowego (sinusoidalny, prostokątny, trójkątny),
- Ustawienie częstotliwości, fazy, czasu i kroku symulacji,
- Wizualizację sygnału wejściowego oraz odpowiedzi układu.
- Sprawdza stabilność układu i wyświetla o niej komunikaty w terminalu.

4 Opis działania programu

```
Gp_licznik = [a1, a0]
Gp_mianownik = [b2, b1, b0]
Gp = ct.tf(Gp_licznik, Gp_mianownik)
Gc_licznik = [Kp, Ki]
Gc_mianownik = [1, 0]


Gc = ct.tf(Gc_licznik, Gc_mianownik)
Go = ct.series(Gc, Gp)
Gz = ct.feedback(Go, 1)
```

Rysunek 1: Po przekazaniu do funkcji run_simulation są tworzone poszczególne transmitancje, są łączone szeregowo i zamykane w ujemnej pętli sprzężenia zwrotnego. W ten sposób otrzymujemy transmitancję całego układu.

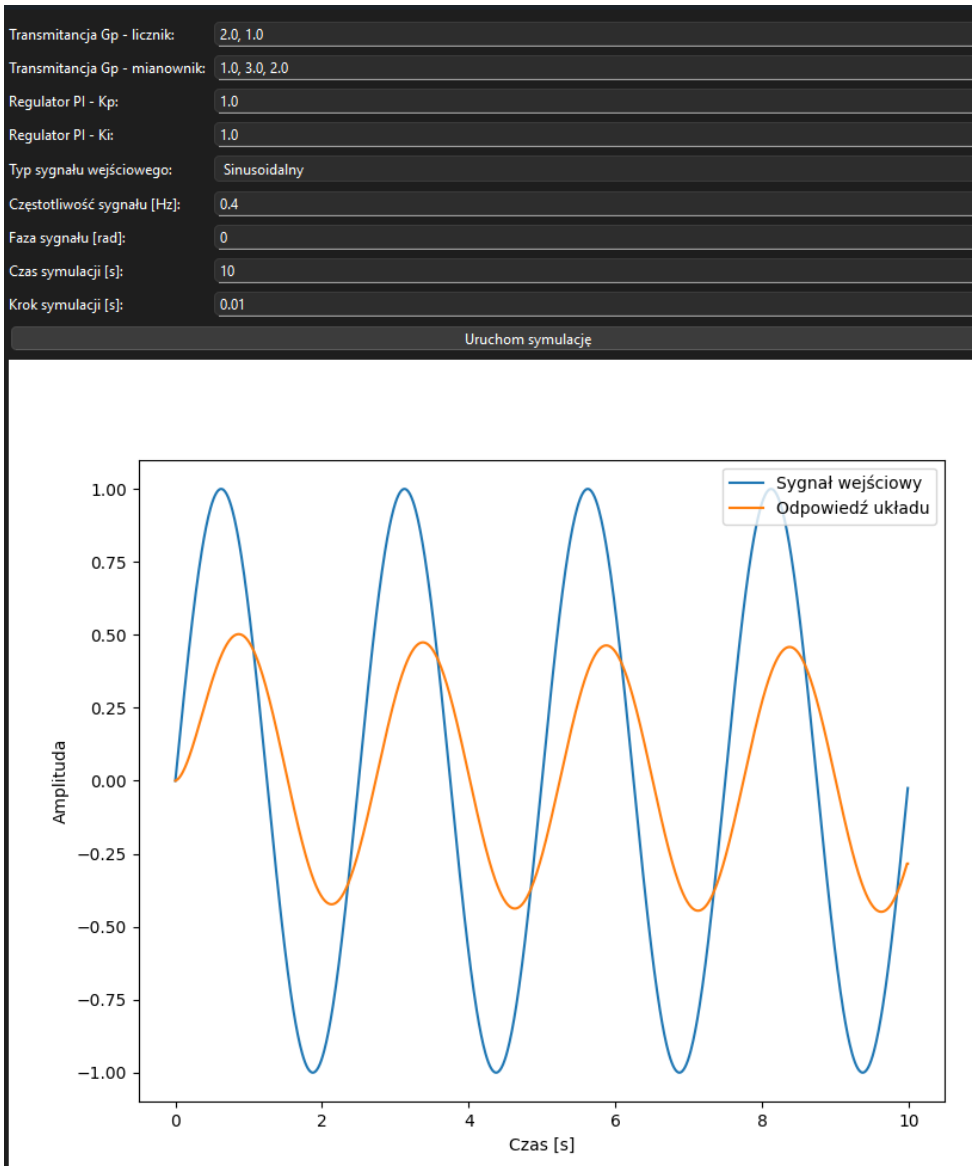
```
ss_model = ct.tf2ss(Gz)
A, B, C, D = ss_model.A, ss_model.B, ss_model.C, ss_model.D
```

Rysunek 2: Transmitancja całego układu jest następnie przekształcana na model stanowy.

```
for i in range(num_elements-1):
    #krok próbny
    Xp = Xp + A @ Xp + B * U[i]
    #krok właściwy
    X = Xp + 0.5 * (A @ X + B * U[i]  + A @ Xp + B * U[i+1])
    Y = C @ X + D * U[i]
    output_signal.append(Y.item())

# Dodanie ostatniego punktu do output_signal
Y = C @ X + D * input_signal[-1]
output_signal.append(Y.item())
```

Rysunek 3: Rozwiązywanie równań stanu jest realizowane przy pomocy metody Tustina.

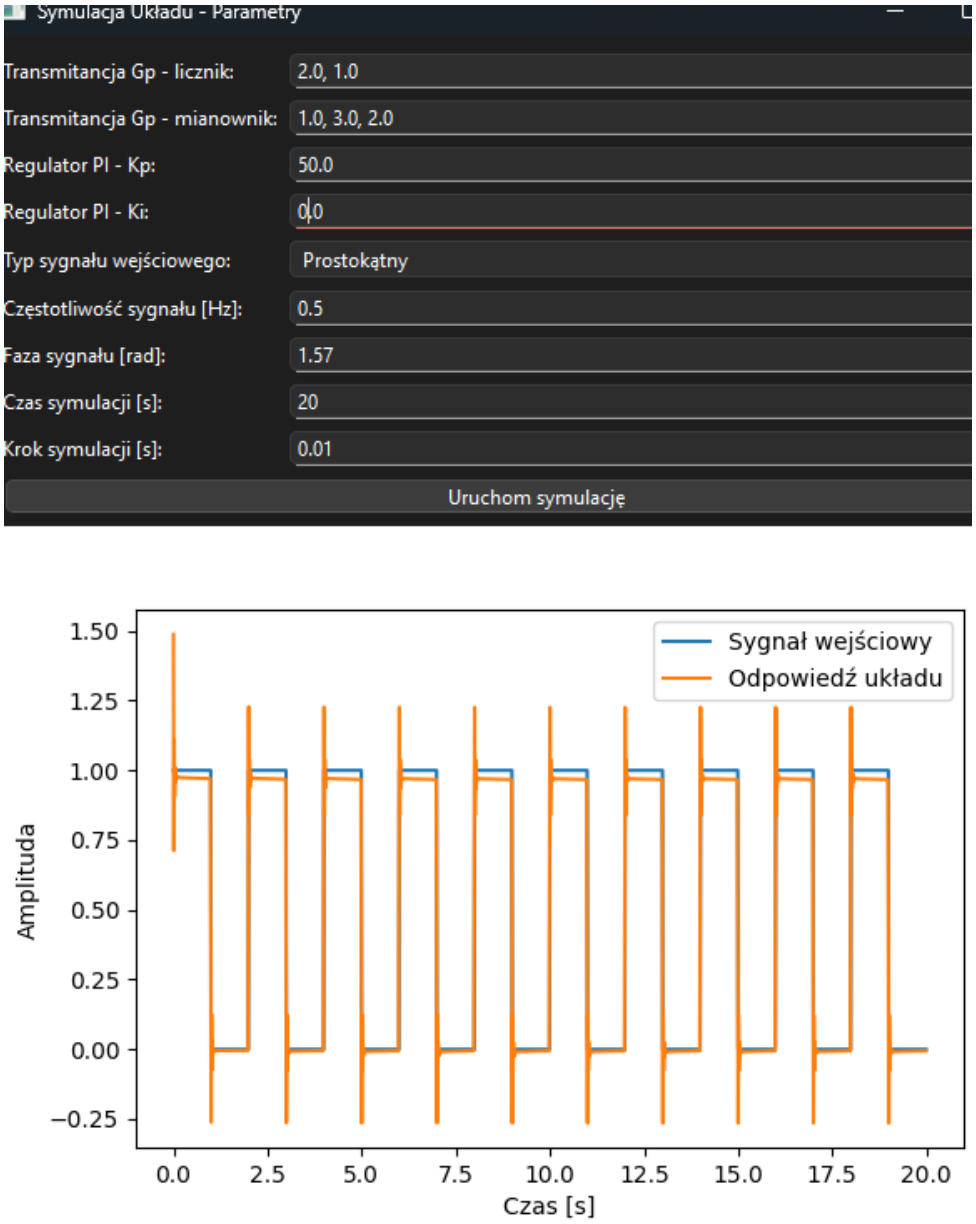


Rysunek 4: Przykładowa odpowiedź układu na sygnał sinusoidalny.

5 Przykładowe wyniki

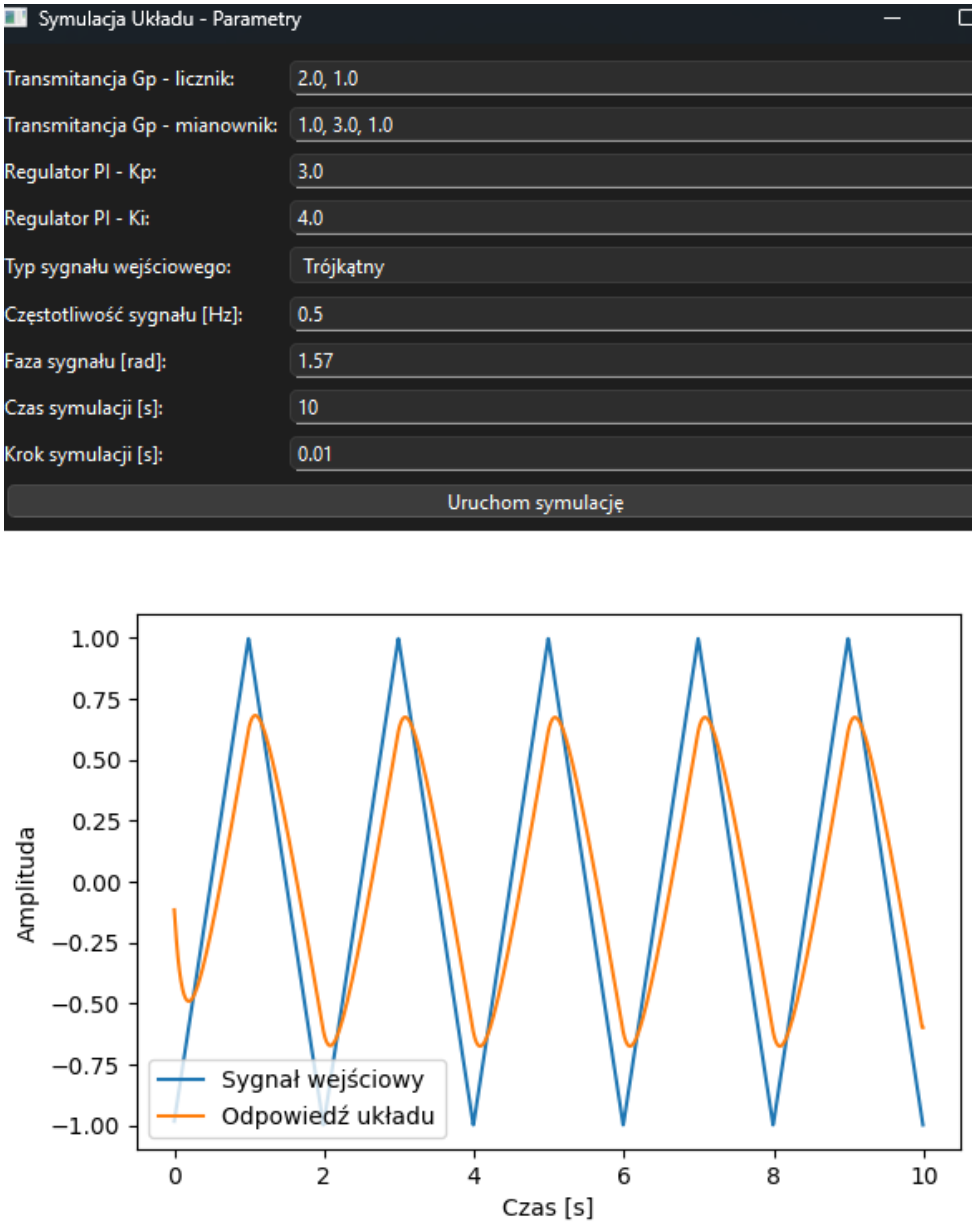
6 Wnioski

Wykorzystanie biblioteki Numpy oraz funkcji sawtooth oraz square z biblioteki SciPy pozwoliła na łatwą implementację sygnałów wejściowych. Użycie biblioteki control pozwoliło na proste utworzenie transmitancji badanego układu oraz przekształcenie jej do modelu stanowego. W zasadzie jedyne ograniczenie w tworzeniu transmitancji obiektu i sterownika leży aktualnie w liczbie argumentów przyjmowanych przez funkcje run_simulation (co można łatwo zmienić). Dodatkowo fakt wykorzystania metody Tustina do rozwiązywania równań stanu pozwala na implementację innych sygnałów wejściowych, które posiadają miejsca nieciągłości(tak jak np. w przypadku sygnału prostokątnego) co byłoby problematyczne w przypadku zastosowania np, metody Taylora. Zastosowanie modelu stanowego umożliwia również na łatwe rozszerzenie projektu o dodatkowe funkcjonalności jak np. dodanie elementów nieliniowych lub liczenie czy wykreślanie również sygnałów wewnętrz-



Rysunek 5: Przykładowa odpowiedź układu na sygnał prostokątny. Ilustracja przeregulowania.

nych układu(oczywiście wymagało by to przebudowy pętli realizującej symulacje tak aby liczyła stan każdego elementu układu).



Rysunek 6: Przykładowa odpowiedź układu na sygnał trójkątny.