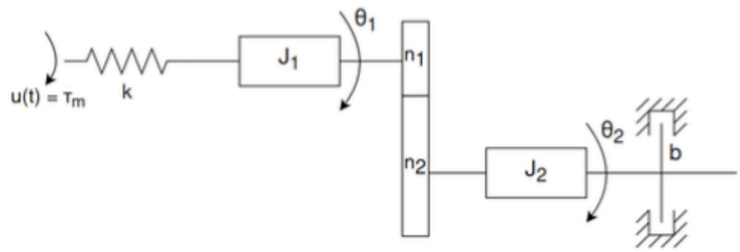
Projekt MMM

Aleksander Fuks 188554
Marcel Czerwiński 188962

Zadanie:

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

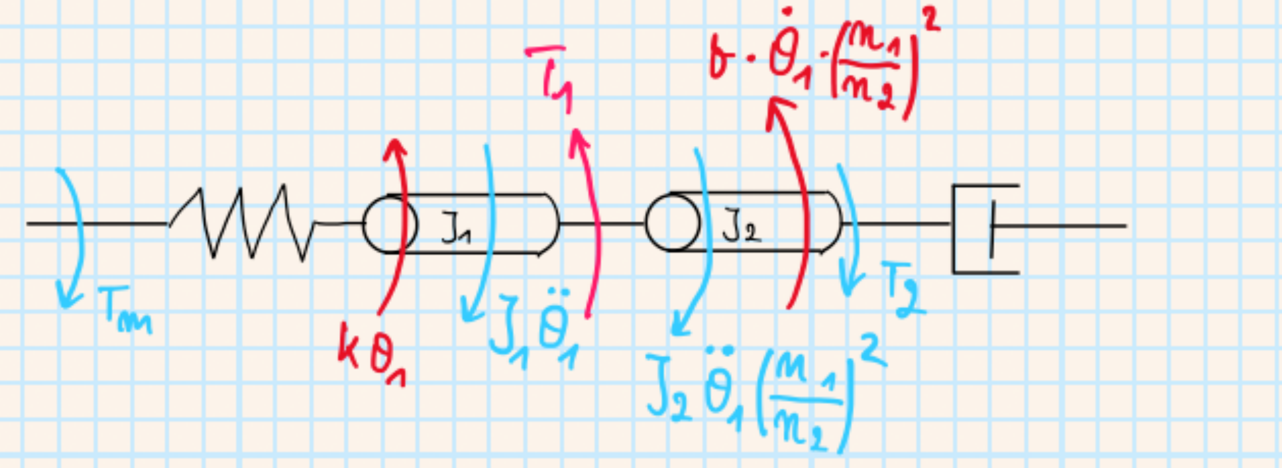


Należy wyprowadzić model układu oraz zaimplementować go w symulacji. Symulator powinien umożliwiać pobudzenie układu przynajmniej trzema rodzajami synagłów wejściowych (prostokątny o skończonym czasie trwania, trójkątny, harmoniczny). Symulator powinien umożliwiać zmianę wszystkich parametrów układu 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.

Gdzie:
Tm - sygnał wejściowy
K-współczynnik sprężystości [N*m/rad]
$J_{1,2}$ - momenty bezwładności [kg*m^2]
b - współczynnik tłumienia
$n_{1,2}$ - liczba zębów przekładni
$\theta_{1,2}$ - kąt obrotu [rad]

Wyliczenie modelu stanowego:

$$\theta_2 = \frac{n_1}{n_2} \cdot \theta_1 \qquad \theta_2 \cdot n_2 = \theta_1 \cdot n_1$$



$$J_2 \ddot{\theta}_2 \frac{n_2}{n_1} + b \cdot \dot{\theta}_2 \frac{n_2}{n_1} = \left(-J_1 \ddot{\theta}_2 \cdot \frac{n_1}{n_2} - k \cdot \frac{n_1}{n_2} \theta_2 + T_m\right) \frac{n_2}{n_1}$$

$$J_1 \ddot{\theta}_2 + J_2 \ddot{\theta}_2 \frac{n_2}{n_1} + b \cdot \dot{\theta}_2 \frac{n_2}{n_1} = -k \cdot \theta_2 + T_m \cdot \frac{n_2}{n_1}$$

$$\ddot{\theta}_2 \left(J_1 + J_2 \frac{n_2}{n_1}\right) = -b \cdot \dot{\theta}_2 \frac{n_2}{n_1} - k \cdot \theta_2 + T_m \cdot \frac{n_2}{n_1}$$

$$\ddot{\theta}_2 = \frac{-b \cdot \dot{\theta}_2 \frac{n_2}{n_1} - k \cdot \theta_2 + T_m \cdot \frac{n_2}{n_1}}{J_1 + J_2 \frac{n_2}{n_1}}$$

$$x_1 = \theta_2 \qquad x_2 = \dot{\theta}_2 \quad \Rightarrow \quad \dot{x}_2 = \frac{-b x_2 \frac{n_2}{n_1} - k \cdot x_1 + T_m \cdot \frac{n_2}{n_1}}{J_1 + J_2 \frac{n_2}{n_1}}$$

$$\dot{x}_1 = x_2$$

$$y = x_1$$

Powyższe równania stanowe zostały użyte w algorytmie Runge'go-Kutty 4-go stopnia oraz w algorytmie Eulera.

Projekt został napisany w języku Python.

Kod:

```
#funkcja przypisująca podane parametry sygnału do odpowiednich zmiennych
def change_signal(entryp, entrya, entryf, entryt, entryhh):
    global phi, amp, frq, czasTrwania, h
    if isfloat(entryp.get()):
        phi = float(entryp.get())
    if isfloat(entrya.get()):
        amp = float(entrya.get())
    if isfloat(entryf.get()):
        frq = float(entryf.get())
    if isfloat(entryt.get()):
        czasTrwania = float(entryt.get())
    if isfloat(entryhh.get()):
        h = float(entryhh.get())
```

Funkcja change_signal pobiera wartości z odpowiednich entryboxów i przypisuje je do globalnych zmiennych jeśli są wartościami zmiennoprzecinkowymi (separatorem dziesiętnym musi być kropka).

```
#fukcje tworzące sygnał wejściowy. Odpowiednio: sinusoidalny, prostokątny i trójkątny
def wakeUpWithSin(aktualnyczas):
    tm = np.sin(2 * np.pi * frq * aktualnyczas + phi)
    return tm


def wakeUpWithRec(aktualnyczas):
    global amp
    value = 0
    tm = np.sin(2 * np.pi * frq * aktualnyczas + phi)
    if tm >= 0:
        value = amp
    else:
        value = 0
    return value


def wakeUpWithTri(aktualnyczas):
    tm = np.arcsin(np.sin(2 * np.pi * frq * aktualnyczas + phi)) * 2 / np.pi
    return tm
```

Tworzenie sygnałów zostało oparte o sinusa z biblioteki numpy. Sygnał prostokątny przyjmuje wartość 1 dla dodatnich wartości sinusa i 0 dla ujemnych. Sygnał trójkątny to zmieniający się w czasie sinus, na który została użyta funkcja arcsin, żeby liniowo ograniczyć wartości od 1 do -1.

```python
# pobieranie danych z okienka i przypisanie do odpowiedniej zmiennej
def checkData(someEntry):
    global licznik, licznik2
    wejscie = someEntry.get()
    if wejscie != "":
        if isfloat(wejscie):
            licznik2 += 1
        elif licznik == 0:
            tkinter.messagebox.showerror(title="Podana wartość nie jest liczbą.",
                                         message="Sprawdź czy użyłeś kropki dziesiętnej")
            licznik = 1
```

```python
def checkczywszytstkogit():
    global licznik2
    if licznik2 == 6:
        tkinter.messagebox.showinfo(message="Wartości zapisane poprawnie")
```

Checkdata sprawdza czy podana przez użytkownika wartość parametru układu jest liczbą zmiennoprzecinkową. Funkcja jest wywoływana w jednym przycisku sześciokrotnie. Zmienna licznik ustawia się na 1, gdy któraś z wartości została błędnie wpisana dzięki czemu wiadomość o niepoprawnym wprowadzeniu wartości wyświetla się jednokrotnie (a nie sześciokrotnie). Zmienna licznik2 zwiększa się o 1 za każdym razem, gdy podana wartość jest prawidłowa, następnie wykonana zostaje funkcja checkczywszystkogit, która informuje użytkownika o poprawnym wprowadzeniu danych tylko i wyłącznie, gdy WSZYSTKIE dane są poprawne. Oba liczniki zerują się po kliknięciu przycisku.

```python
# pobieranie danych z okienka i przypisanie do odpowiedniej zmiennej
def checkSignal(someEntry):
    global licznik3, licznik4
    wejscie = someEntry.get()
    if wejscie != "":
        if isfloat(wejscie):
            licznik4 += 1
        elif licznik3 == 0:
            tkinter.messagebox.showerror(title="Podana wartość nie jest liczbą.",
                                         message="Sprawdź czy użyłeś kropki dziesiętnej")
            licznik3 = 1
```

```python
def checkczySignalgit():
    global licznik4
    if licznik4 == 4:
        tkinter.messagebox.showinfo(message="Wartości zapisane poprawnie")
```

Podobnie jak wyżej, ale funkcje dotyczą zmian sygnału wejściowego. Zostały napisane dwie osobne funkcje, ponieważ zaimplementowaliśmy dwa osobne przyciski do zmiany sygnału i zmiany parametrów układu.

```python
def rownanienax1(x2poprzedni):
    x1prim = x2poprzedni
    return x1prim


def rownanienax2(x2poprzedni, x1poprzedni, aktualnyczas, option):
    global amp
    value = option.get()
    if value == "sine":
        x2prim = (-b * x2poprzedni * n2 / n1 - k * x1poprzedni + wakeUpWithSin(aktualnyczas) * amp * n2 / n1) / (
                j1 + j2 * n2 / n1)
    elif value == "rectangle":
        x2prim = (-b * x2poprzedni * n2 / n1 - k * x1poprzedni + wakeUpWithRec(aktualnyczas) * n2 / n1) / (
                j1 + j2 * n2 / n1)
    elif value == "triangle":
        x2prim = (-b * x2poprzedni * n2 / n1 - k * x1poprzedni + wakeUpWithTri(aktualnyczas) * amp * n2 / n1) / (
                j1 + j2 * n2 / n1)

    return x2prim
```

Powyższe funkcje są wykorzystywane w algorytmach do liczenia kolejnych wartości x1 i x2. W zależności od wyboru rodzaju sygnału przez użytkownika zostaje użyta odpowiednia funkcja do liczenia x2. Wielkość zwracana przez funkcję wakeUpWithRec nie jest mnożona przez amp, ponieważ zmiana amplitudy tego sygnału została zaimplementowana wcześniej. Wybór rodzaju sygnału (sin, rec, tri) jest realizowany poprzez przyciski radiowe.

```python
def RungeKutta4stopnia(option):
    x1.clear()
    x2.clear()
    t.clear()
    x1.append(0)
    x2.append(0)
    t.append(0)
    tnowy = 0

    #liczenie kolejnych wartości x2 i x1
    for i in range(1, int(czasTrwania / h)):
        k1 = h * rownanienax2(x2[i - 1], x1[i - 1], t[i - 1], option)
        l1 = h * rownanienax1(x2[i - 1])

        k2 = h * rownanienax2(x2[i - 1]+k1/2, x1[i - 1]+h/2, t[i - 1], option)
        l2 = h * rownanienax1(x2[i - 1]+h/2)

        k3 = h * rownanienax2(x2[i - 1]+k2/2, x1[i - 1]+h/2, t[i - 1], option)
        l3 = h * rownanienax1(x2[i - 1]+h/2)

        k4 = h * rownanienax2(x2[i - 1]+k3, x1[i - 1]+h, t[i - 1], option)
        l4 = h * rownanienax1(x2[i - 1]+h)

        x2nowy = x2[i - 1] + 1 / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
        x2.append(x2nowy)
        x1nowy = x1[i - 1] + 1 / 6 * (l1 + 2 * l2 + 2 * l3 + l4)
        x1.append(x1nowy)
        tnowy = tnowy + h
        t.append(tnowy)
```

Czyszczone zostają listy z danymi oraz wprowadzone zostaje 0 do każdej listy, ponieważ kolejne wartości x1, x2, t są liczone na podstawie poprzednich. Czas zmienia się liniowo o stałą całkowania i ilość obliczeń zależy od wprowadzonego przez użytkownika czasu.

```python
def euler(option):
    x1euler.clear()
    x2euler.clear()
    x1euler.append(0)
    x2euler.append(0)
    t.clear()
    t.append(0)
    tnowy = 0

    for i in range(1, int(czasTrwania / h)):
        k1 = h * rownanienax2(x2euler[i - 1], x1euler[i - 1], t[i - 1], option)
        l1 = h * rownanienax1(x2euler[i - 1])

        x2nowye = x2euler[i - 1] + k1
        x2euler.append(x2nowye)
        x1nowye = x1euler[i - 1] + l1
        x1euler.append(x1nowye)
        tnowy = tnowy + h
        t.append(tnowy)
```

Jak wyżej.

```python
#wykreślanie charakterystyk
def rysujx2x1():
    fig, axs = plt.subplots(2)
    fig.suptitle('Project no.4')
    axs[0].plot(t, x1, c='b', label="Angle RK 4th order")
    axs[0].plot(t, x1euler, c='r', label="Angle Euler")
    axs[0].legend(loc='upper right')

    axs[1].plot(t, x2, c='b', label="Angular velocity RK 4th order")
    axs[1].plot(t, x2euler, c='r', label= "Angular velocity Euler")
    axs[1].legend(loc='upper right')

    plt.show()
```

Narysowane zostają dwie funkcje jedna pod drugą. axs odpowiada za wybór wiersza, w którym odpowiednia funkcja się rysuje.

```python
#przyciski
changeentries = tk.Button(
    text="Change model's parameters' values", width=24, height=1, bg="gold", fg="black", padx="1", pady="1",
    command=lambda: [changelicznik(), changelicznik2(), checkData(entryn1),
                     checkData(entryn2), checkData(entryj1), checkData(entryj2),
                     checkData(entryk), checkData(entryb),
                     changeParam(entryn1, entryn2, entryj1, entryj2, entryb, entryk), checkczywszytstkogit()],
)
changeentries.place(x=10, y=310)
```

Po wciśnięciu wyzerowane zostają liczniki, sprawdzona zostaje poprawność wpisanych danych oraz następuje zmiana parametrów. Wprowadzanie jednej lub kilku błędnej danej nie skutkuje zresetowaniem pozostałych okienek. Poprawnie wpisane wartości zostaną zapisane i możliwe będzie wykonanie symulacji.

```python
signal = tk.Button(
    text="Change signal ap rameters", width=20, height=2, bg="black", fg="red", padx="1", pady="1",
    command=lambda: [changesignalliczniki(), checkSignal(entrytime), checkSignal(entryphase), checkSignal(entryamp),
    checkSignal(entryfrq), change_signal(entryphase, entryamp, entryfrq, entrytime, entryh), checkczySignalgit()])
signal.place(x=160, y=270)
```

Jak wyżej, ale do sygnału.

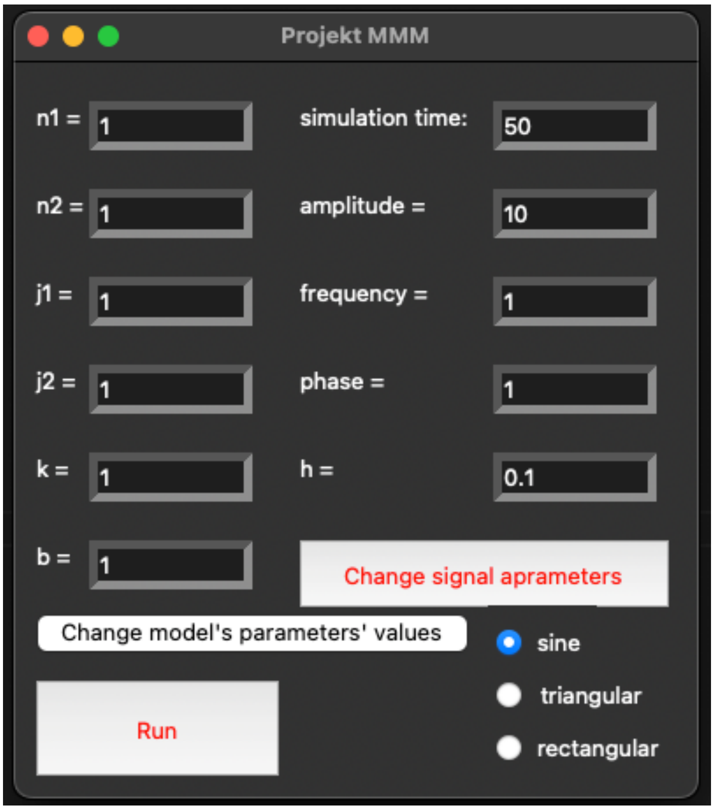
```python
run = tk.Button(
    text="Run", width=12, height=3, bg="black", fg="red", padx="1", pady="1",
    command=lambda: [RungeKutta4stopnia(opcja), euler(opcja), rysujx2x1()])
run.place(x=10, y=350)
```

Zaimplementowany zostaje algorytm Rungego Kutty oraz Eulera, następnie na podstawie tych danych rysowane są wykresy.

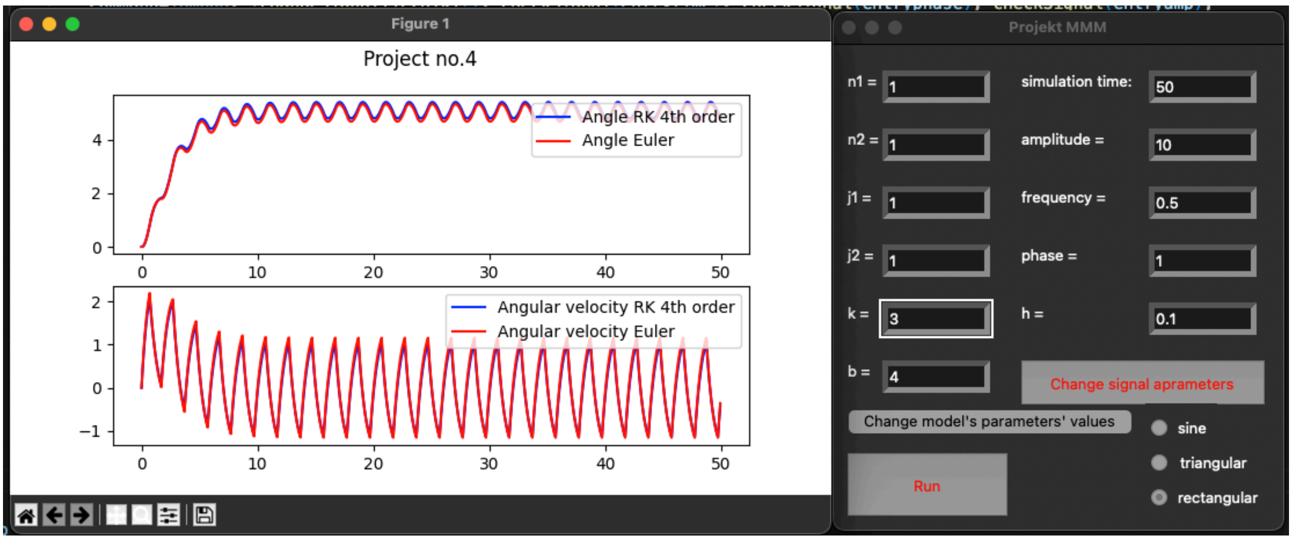
```python
sine = tk.Radiobutton(
    text="sine", width=6, height=2, variable=opcja, value="sine",
)
sine.place(x=270, y=310)
sine.invoke()

triangle = tk.Radiobutton(
    text="triangular", width=10, height=2, variable=opcja, value="triangle",
)
triangle.place(x=270, y=340)

rectangle = tk.Radiobutton(
    text="rectangular", width=11, height=2, variable=opcja, value="rectangle",
)
rectangle.place(x=270, y=370)
```

Wybór rodzaju pobudzenia, domyślnie jest to sygnał sinusoidalny. Wartości tych przycisków wykorzystywane są w funkcji RungeKutta4stopnia i euler.

Interfejs:



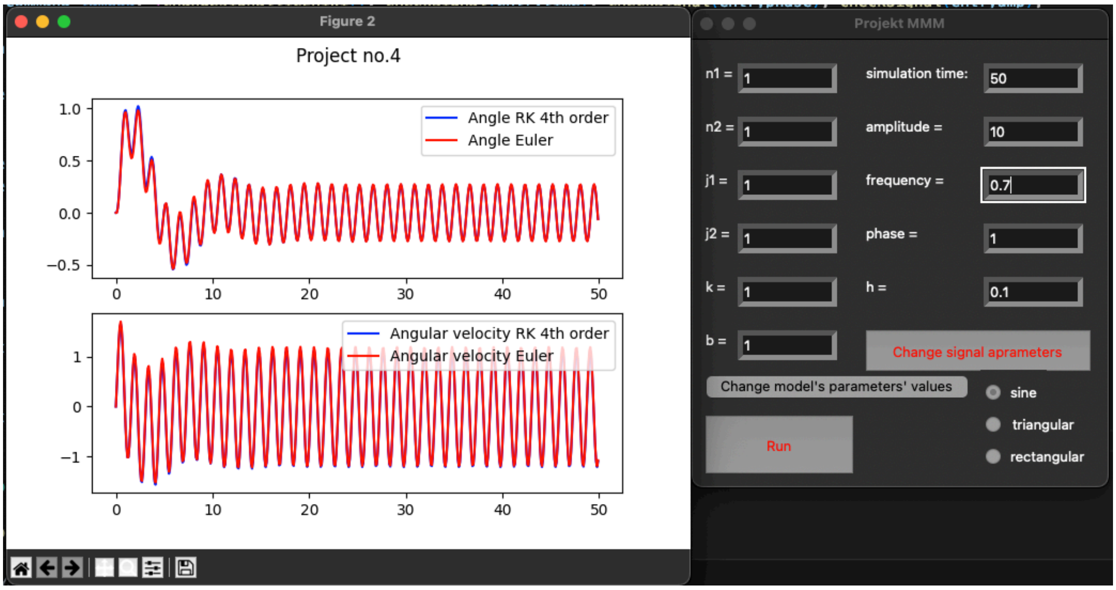
Program z domyślnymi wartościami zmiennych.



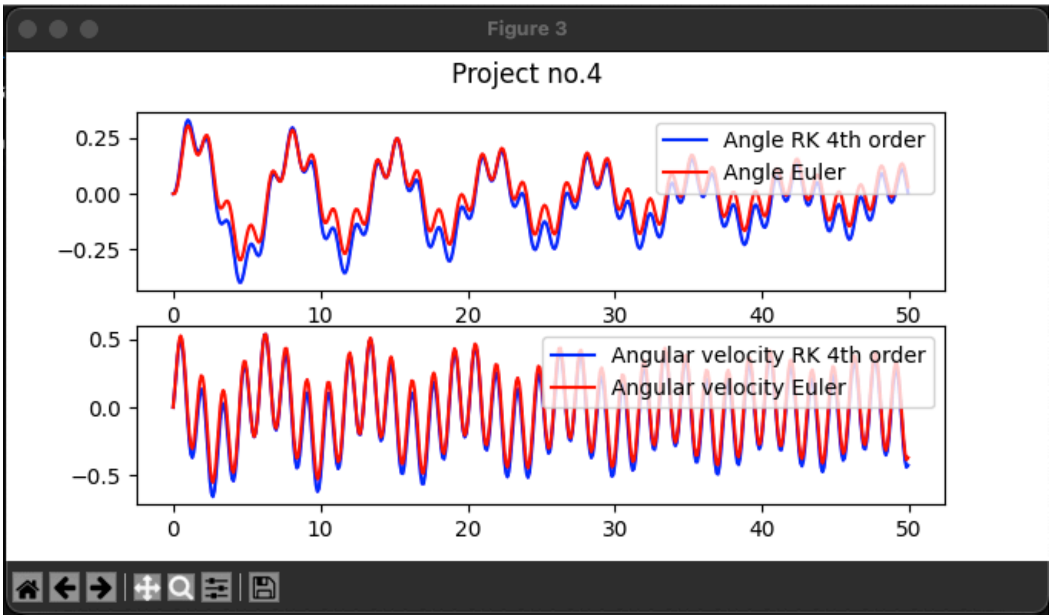
Przykład działania programu.

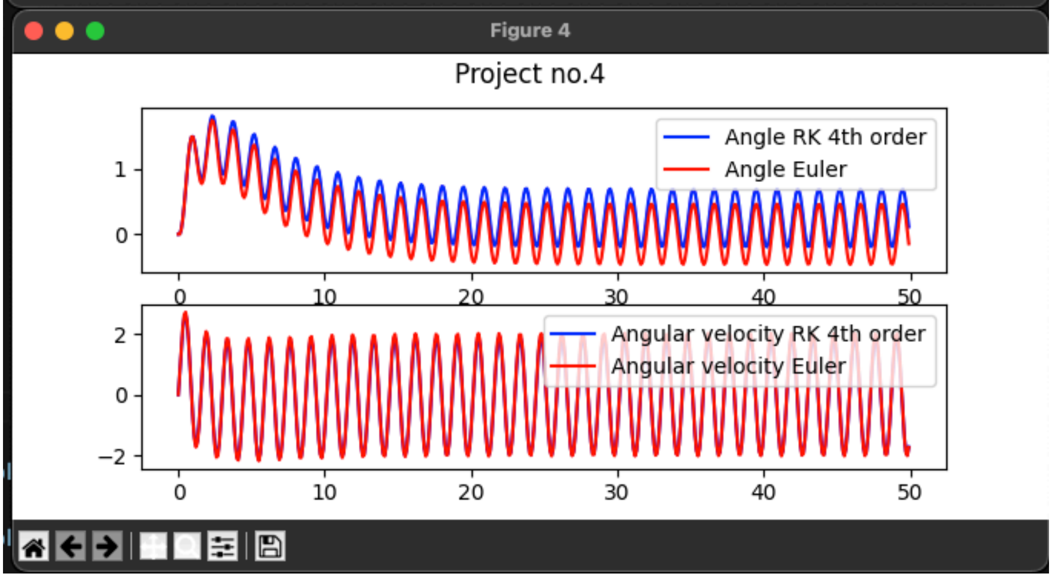
Symulacje:

Zmieniane zostają pojedynczo parametry modelu, pozostałe parametry przyjmują wartość bazową.

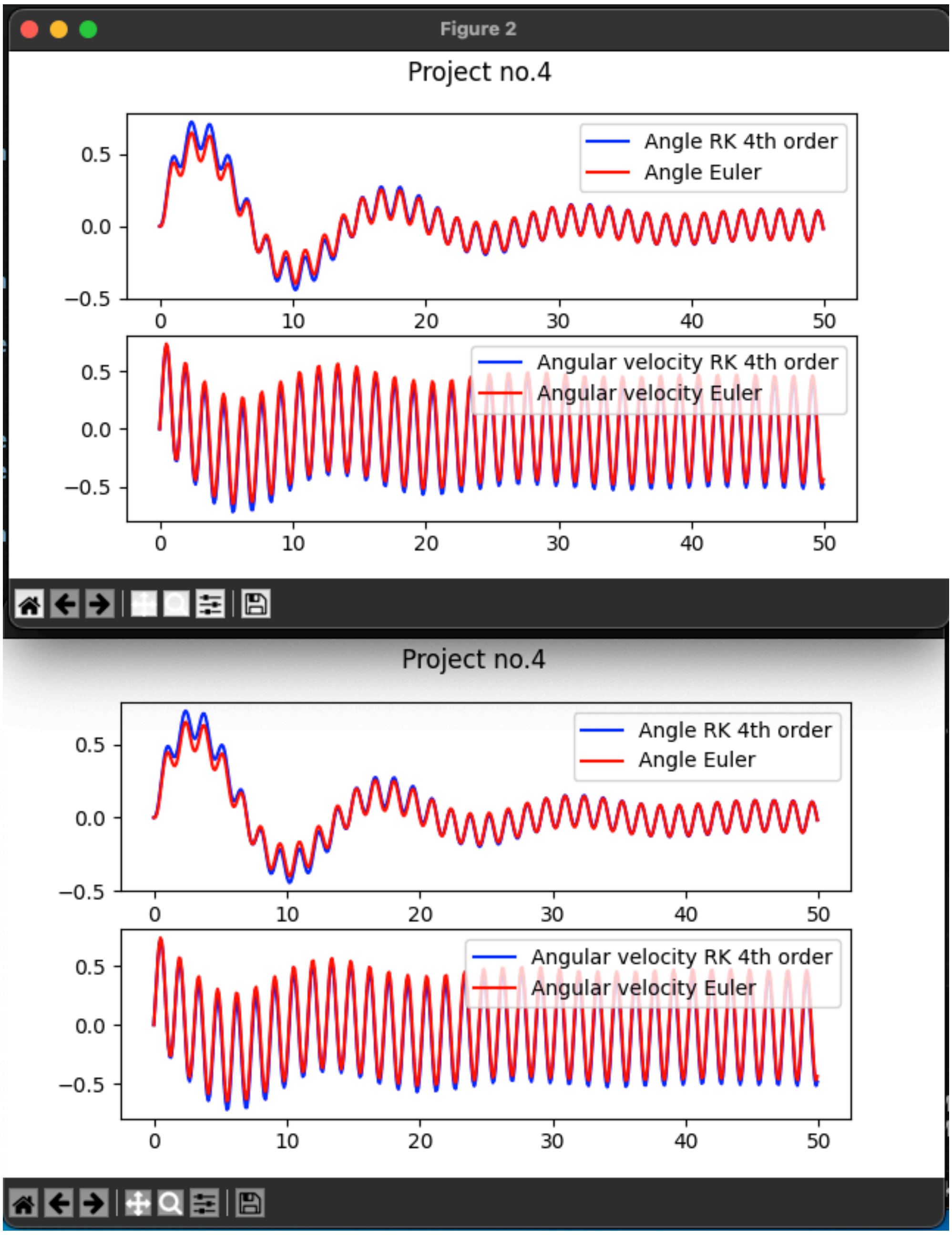


Symulacja z bazowymi wartościami parametrów, z którą będziemy porównywać kolejne symulacje.

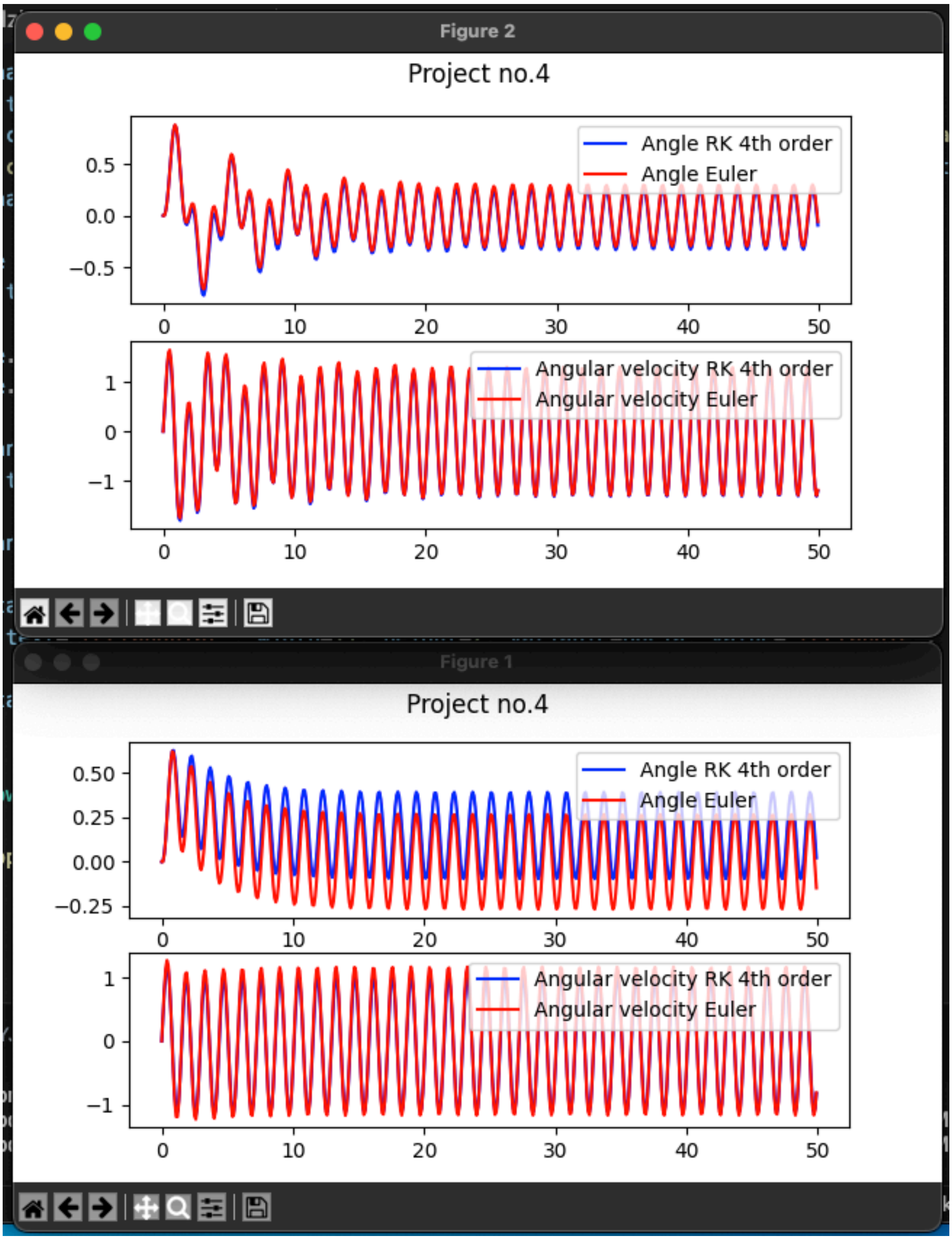




Na wykresie górnym stosunek n1/n2 wynosi 6, a na dolnym 1/6. Większa zębatka dolna powoduje zmniejszenie pracy potrzebnej do płynnego obrotu wałem przez co stan ustala się szybciej.



Wykreślone zostały sygnały ze zmienionym odpowiednio J1 i J2 (u góry J1 = 4, J2 = 1, a na dole odwrotnie). Wykresy są identyczne, co jest zgodne z oczekiwaniami, ponieważ stosunek n2/n1 jest równy 1, czyli w tym przypadku zmiana J1 wpływa tak samo na układ jak zmiana J2. Układ potrzebuje więcej czasu na stabilizację oraz drgania mają mniejszą amplitudę. Jest to zgodne z założeniami - został zwiększony moment bezwładności.



Na wykresie górnym został zwiększony parametr k, a na dolnym b. Odpowiedzi są zgodne z założeniami - po zwiększeniu k występuje więcej przeregulowań, a przy większym b przeregulowania nie występują.

Wartości

Podsumowanie:

Symulacje pokazują charakterystyki zgodne z rzeczywistymi założeniami. Symulacje przeprowadzane metodą Eulera dają mniej dokładne wyniki. Objawia się to stabilizacją sygnału na innym poziomie. Przeprowadzone symulacje wskazują na stabilność układu w sensie BIBO.