Układy Sterowania Optymalnego

Politechnika Poznańska
Instytut Automatyki i Robotyki

ĆWICZENIE 1

Wprowadzenie do języka Python

Celem ćwiczenia jest zaznajomienie z programowaniem w języku Python. Przedstawione zostaną podstawy składni i semantyki języka oraz najpopularniejsze biblioteki wykorzystywane w praktyce inżynierskiej oraz badawczej.

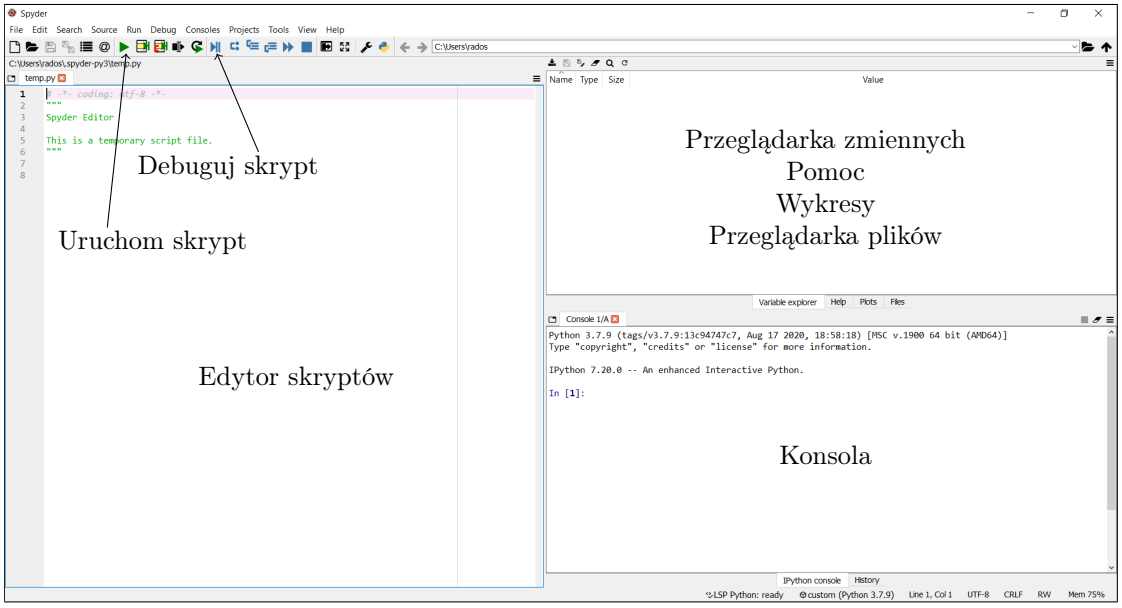
W ramach przygotowania do ćwiczenia należy:

→ Przypomnieć wiadomości z zakresu:

- programowania w językach ogólnego przeznaczenia (np. C, C++, Java)

1 Środowisko Spyder

W ramach zajęć wykorzystywane będzie środowisko Spyder (https://www.spyder-ide.org/) przystosowane do prac badawczych i rozwojowych. Zarówno pod kątem wizualnym jak i funkcjonalnym jest ono wzorowane na pakiecie Matlab. Środowisko dostępne jest za darmo na większości popularnych systemów operacyjnych (Windows, MacOS, Linux, itp.) i zawiera w sobie dystrybucję Python wraz z zestawem pakietów naukowych.



Rysunek 1: GUI środowiska Spyder

Interfejs użytkownika przedstawiony na Rys. 1 jest silnie konfigurowalny, natomiast domyślnie podzielony został na trzy główne panele:

- Edytor skryptów (z lewej) - pozwala na edycję i zapisywanie skryptów, przyciski na górnej belce pozwalają uruchomić i debugować przygotowany program,
- Konsola (z prawej na dole) - pozwala na jednorazowe wywołanie dowolnych wyrażeń, np. tworzenie zmiennych w przestrzeni roboczej,
- Panel wielofunkcyjny (z prawej na górze) - poprzez zakładki pozwala na wybranie jednej z kilku, dostępnych funkcji
  - Przeglądarka zmiennych - wyświetla zdefiniowane aktualnie zmienne (utworzone poprzez konsolę lub uruchamiane wcześniej skrypty),
  - Pomoc - pokazuje dokumentację wybranej funkcji lub klasy, by wyświetlić dokumentację należy zaznaczyć wyrażenie i wcisnąć CTRL+i,
  - Okno wykresów - przechowuje wszystkie wygenerowane dotychczas wykresy. Przyciski na górze panelu pozwalają wyczyścić pamięć z niepotrzebnych już wyników,
  - Przeglądarka plików - podgląd zawartości aktualnego folderu roboczego. Pozwala otwierać pliki w edytorze skryptów.

2 Typy danych w Pythonie

Składnia Pythona jest zbliżona do składni innych popularnych języków ogólnego przeznaczenia - w wielu kwestiach została jednak uproszczona dla przyspieszenia i ułatwienia zarówno pisania jak i odczytywania wcześniej przygotowanego kodu. Python wykorzystuje dynamiczne typowanie, zatem przy tworzeniu zmiennej nie jest wymagane wskazania typu przechowywanych danych, a poniższe komendy poprawnie stworzą dwie zmienne przechowujące odpowiednio wartość numeryczną oraz tekstową.

```
a = 13 #zmienna numeryczna
b = 'Hello World' #zmienna tekstowa
```

Python nie wymaga (chociaż zezwala) także kończenia komend średnikiem - przejście do nowej linii jest jednoznaczne w zakończeniem danego polecenia. W celu zapisania komendy złożonej z kilku linii kodu należy wykorzystać znak blackslash (\). Komentarze oznaczane są znakiem kratki (#). Podobnie jak w innych językach możliwe jest utworzenie i odwoływanie się do tablic zmiennych. **Tablice w Pythone indeksowane są od wartości 0 (inaczej niż w Matlabie!)**

```
a = [0,2,4,6,8,10]
a[0] = 2
a[2] = 6
a[5] = 8
```

Powyższy kod skutkuje utworzeniem tablicy zawierającej ciąg `[2,2,6,6,8,8]`. Dodatkowe biblioteki umożliwiają korzystanie z bardziej złożonych typów, np. macierzy czy wektorów. Poniższy kod załącza bibliotekę `numpy` zawierającą kolekcję najważniejszych struktur i funkcji matematycznych, po czym tworzy obiekty reprezentujące jednowymiarowy wektor, macierz dwuwymiarową oraz wektor w przestrzeni dwuwymiarowej

```
import numpy as np
a = np.array([1,3,5,7]) #wektor 1D
b = np.array([[1,2],[3,4]]) #macierz 2D
c = np.array([[1,2,3,4]]) #wektor 2D
d = np.matrix([[1,2],[3,4]]) #takze macierz 2D
c = np.matrix([[1,2,3,4]]) #takze wektor 2D
```

Należy pamiętać, że do elementów macierzy i wektorów 2D odwoływać się trzeba z wykorzystaniem dwóch indeksów

```
c = np.array([[1,2,3,4]])
c[0][0] = -1
c[0][1] = -2
```

Biblioteka `numpy` pozwala także na wygodniejsze generowanie tablic, np. poprzez określenie zakresu który ma obejmować tablica. Poniższy kod tworzy trzy takie same tablice poprzez podanie wprost elementów, wskazanie odstępów między elementami lub wskazanie pożądanej liczby elementów.

```
import numpy as np
a = np.array([0,2,4,8,10])
b = np.arange(0,10,2)
c = np.linspace(0,5,10)
```

Zarówno na typach podstawowych jak i na typach definiowanych przez `numpy` możliwe jest wykonywanie operacji algebraicznych. Oprócz podstawowych operacji dodawania, odejmowania, mnożenia i dzielenia dostępny jest także operator potęgowania (operator `**`, potęgowanie elementów dla `numpy.array` oraz potęgowanie macierzy dla `numpy.matrix`!) oraz mnożenia macierzowego (tylko obiekty `numpy`, operator `@`). Korzystając z mnożenia macierzowego istotne jest zachowanie takiej samej wymiarowości wszystkich czynników (tj. należy mnożyć macierz 2d razy wektor 2d, a nie macierz 2d razy wektor 1d). Dla operacji na macierzach istotne jest także zachowanie dopuszczalnej kolejności działań. Poniżej przedstawiono przykładowe zastosowanie podstawowych operatorów matematycznych.

```
a = 4
b = 3
c = a+b
d = a**b
```

```
import numpy as np
e = np.array([[1,2,3],[4,5,6],[7,8,9]])
f = np.array([[1,3,5]])
g = np.array([[2],[4],[6]])
x = f @ e
y = e @ g
```

Więcej informacji o operacjach dozwolonych na obiektach `numpy` znaleźć można w dokumentacji numpy (https://numpy.org/doc/).

2.1 Wykorzystując Pythona oraz środowisko Spyder wyznaczyć wartość x. Skorzystać z pakietu algebry liniowej `numpy.linalg`.

- $x = 3^{12} - 5$
- $x = \begin{bmatrix} 2 & 0.5 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} -1 \\ -3 \end{bmatrix}$
- $x = \text{rank}(\begin{bmatrix} 1 & -2 & 0 \\ -2 & 4 & 0 \\ 2 & -1 & 7 \end{bmatrix})$
- $\begin{bmatrix} -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix} x$

2.2 Dana jest tablica $[1, 1, -129, 171, 1620]$. Napisać skrypt tworzący zmienną przechowującą tę tablicę. Przyjmując zawartość tablicy jako współczynniki wielomianu (zaczynając od najwyższej potęgi) wyznaczyć wartość tego wielomianu w punktach $x_1 = -46, x_2 = 14$

3 Pętle i kontrola przepływu

Dla uzyskania wysokiego poziomu czytelności języka, Python posiada tylko dwa rodzaje pętli - pętlę `for` iterującą po wszystkich elementach tablicy (odpowiednik `range for` lub `foreach`) oraz pętlę `while` wykonywaną dopóki prawdziwy jest warunek logiczny. Kod poniżej zawiera trzy pętle, każda z nich zostanie wykonana pięć razy i wypisze do konsoli liczby od 1 do 5 (korzystając z komendy `print`.

```
for x in [1,2,3,4,5]:
    print(x)

import numpy as np
for x in np.array([[1,2,3,4,5]]):
    print(x)

i = 0
while i < 5:
    i += 1
    print(i)
```

Jak można zaobserwować, Python dobrze współpracuje ze złożonymi typami z biblioteki `numpy`. Wszystkie pętle (a także funkcje i inne podobne fragmenty kodu) rozpoczynają się od deklaracji zakończonej dwukropkiem, natomiast blok kodu stanowiący ciało pętli wyznaczany jest poprzez zastosowanie wcięć - każda kolejna linia kodu poprzedzona takim samym wcięciem jak poprzednia należy do tego samego bloku.

Analogicznie do pętli definiuje się operacje warunkowe przy użyciu słowa kluczowego `if`, `else` oraz `elif`

```
a = 1
b = 2
if a > b:
    print(a)
elif b > a:
    print(b)
else:
    exit()
```

Powyższy kod wyświetli większą z dwóch wartości. Jeśli wartości są równe w konsoli program zakończy działanie.

3.1 Bazując na poprzednim zadaniu rozbudować skrypt tak, by w sposób numeryczny (np. poprzez wyznaczanie wartości wielomianu w kolejnych punktach) wyznaczył największą i najmniejszą wartość wielomianu w przedziale $[-46, 14]$.

3.2 Dokładność numerycznego poszukiwania ekstremów zależy od gęstości próbkowania wielomianu - wyznaczając wartość wielomianu w większej ilości punktów zwiększa dokładność obliczeń. Wprowadzić do skryptu dodatkową zmienną pozwalającą ustalić pożądaną dokładność wyznaczania ekstremów.

4 Funkcje

Deklaracja funkcji w Pythonie odbywa się poprzez słowo kluczowe `def`, po którym następuje nazwa funkcji oraz lista argumentów. Ze względu na dynamiczne typowanie nie jest wymagane podawanie informacji o tym, czy funkcja zwraca jakąś wartość. Deklarację przykładowej funkcji oraz sposób ich wywołania przedstawiono poniżej.

```
def my_print(a):
    print(a)
```

```
def my_sum(a,b):
    c = a + b
    return c

x = 1
y = 2
z = my_sum(x,y)
my_print(z)
```

Możliwe jest przypisane domyślnych wartości części lub wszystkich argumentów. Wywołanie funkcji odbywa się poprawnie także przy podaniu jedynie wybranych parametrów.

```
def my_array_creator(a=0,b=0,c=0,d=0):
    return [a,b,c,d]

x = my_array_creator(1, c=4)
y = [1,0,4,0]
```

Powyższy kod stworzył dwie zmienne o takiej samej zawartości - pominięte argumenty funkcji zostały wywołane z wartościami domyślnymi.

4.1 Zmodyfikować skrypt z poprzedniego zadania tak, by funkcjonalność wyznaczająca maksimmum i minimum wielomianu zawarta była w funkcji przyjmującej współczynniki wielomianu 4 stopnia, granice wyznaczania oraz wskaźnik dokładności jako argumenty. Wyznaczone wartości minimum i maksimum powinny być zwracane jako tablica dwuelementowa.

4.2 Rozbudować skrypt, by funkcja przyjmowała jako argument tablicę współczynników wielomianu dowolnego stopnia. W ciele funkcji wykorzystać długość wektora wejściowego do określenia stopnia wielomianu.

5 Wykresy

Python udostępnia rozbudowaną funkcjonalność generowania wykresów dzięki bibliotece matplotlib z pakietem pyplot, którego struktura i składnia silnie wzorowane są na języku Matlab. Poniższy kod generuje wykres funkcji sinus oraz cosinus.

```
import numpy as np
import matplotlib.pyplot as plt

t = np.linspace(0,3*3.14,100)
x = np.sin(t)
y = np.cos(t)

plt.plot(t,x)
plt.plot(t,y)
```

Pyplot domyślnie rysuje przebiegi na tym samym wykresie (odpowiednik `hold on` w Matlabie) jeśli komendy zostały wywołane w skrypcie. By wykreślić przebiegi na jednym wykresie w konsoli należy rozdzielić je kombinacją Enter+Ctrl zamiast przycisku Enter. By wykreślić przebiegi na nowym wykresie zastosować można komendę `plt.figure()` tworzącą nowy wykres.

Wygenerowany wykres można przystosować komendami `plt.xlabel()`, `plt.ymax()`, `plt.legend()`, itp. w sposób analogiczny jak w Matlabie. Finalnie uzyskany wykres można zapisać do pliku

```
import matplotlib.pyplot as plt
x = [1,2,3,2,1,0,1,2,3,2,1,0]

plt.plot(x)
plt.savefig("wykres.pdf", format = 'pdf')
```

Zapis do formatu pdf skutkuje uzyskaniem wektorowej wersji wykresu. Możliwy jest także zapis do formatów rastrowych.

5.1 Zmodyfikować skrypt z poprzedniego zadania tak, by funkcja wyznaczająca ekstrema wielomianu wykreślała przebieg wielomianu w całym wymaganym zakresie.

5.2 Dodać do generowanego wykresu legendę, opisy osi, tytuł, itp.