AKADEMIA GÓRNICZO-HUTNICZA

im. Stanisława Staszica w Krakowie

Wydział Inżynierii Mechanicznej i Robotyki
Katedra Systemów Energetycznych i Urządzeń Ochrony Środowiska

Metody Numeryczne

Laboratorium 1

Wprowadzenie do środowiska obliczeniowego MATLAB

dr inż. Krystian Szopa
KRAKÓW, 2020

1 Środowisko Matlab

Matlab jest programem komputerowym służącym do wykonywania obliczeń inżynierskich i naukowych. Środowisko ma swój język programowania, który umożliwia pisanie własnych skryptów, algorytmów i rozbudowanych programów. Matlab jest szeroko stosowany do przetwarzania i analizy danych pomiarowych. Duże możliwości w zakresie wizualizacji wyników oraz współpraca z zewnętrznymi pakietami, takimi jak LabVIEW, Ansys, sprawia że Matlab jest atrakcyjnym narzędziem dla inżynierów.

W ramach zajęć laboratoryjnych z Metod Numerycznych, Matlab będzie wykorzystywany jako środowisko do realizacji zadań. Niniejszy materiał jest wprowadzeniem do programu Matlab, a informacje w nim zawarte są podstawą kolejnych zajęć.

1.1 Instalacja oprogramowania

Program Matlab jest dostępny dla studentów AGH. Opis instalacji oprogramowania został przedstawiony na stronie:

`http://home.agh.edu.pl/matlab`

Uwaga! W przypadku problemu z instalacją programu Matlab, można zainstalować darmowe (Open Source) środowisko Octave, które w swoich podstawach (język programu), działa podobnie do Matlaba. Opis instalacji Octave zamieszczono na końcu tej instrukcji.

1.2 Interfejs użytkownika

Oprogramowanie zostanie opisane na podstawie wersji Matlab R2020a, jednak zmiany pomiędzy kolejnymi wersjami są niewielkie i z powodzeniem można wykorzystać tę instrukcję do nowszych (i starszych) wersji.

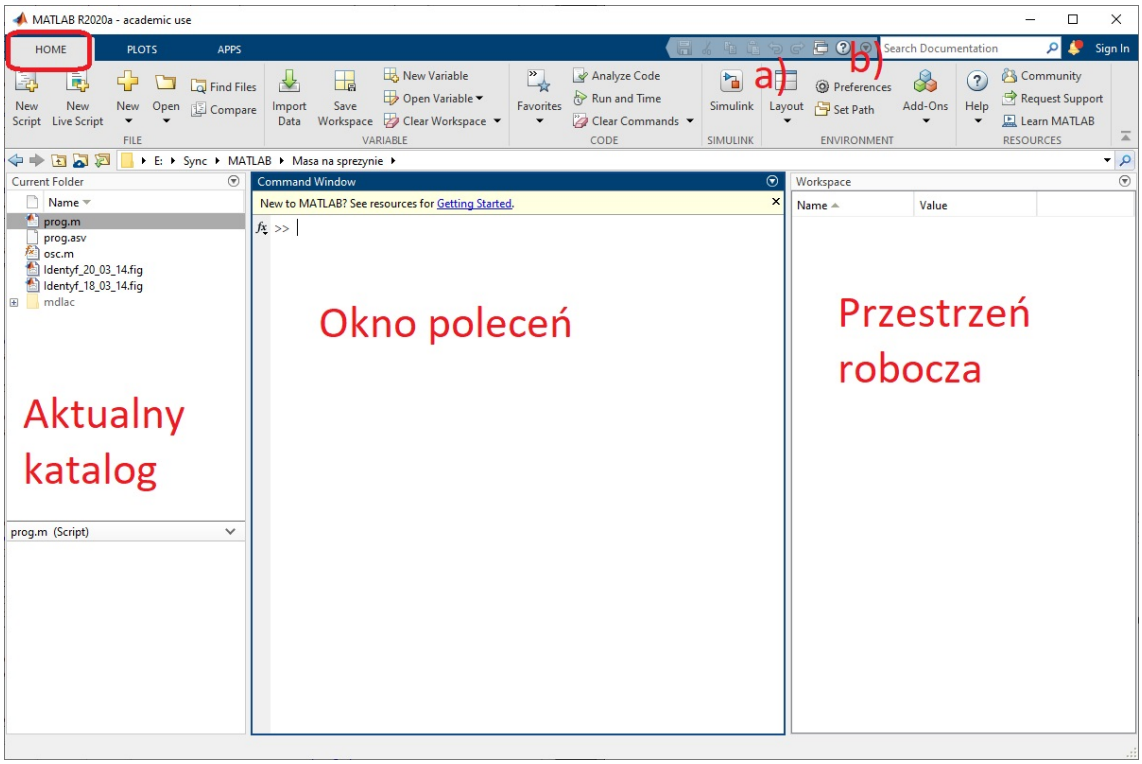
Po uruchomieniu Matlaba, interfejs dla *świeżej* instalacji oprogramowania powinien wyglądać jak na rys. 1.
1. Okno zarządzania - okno poleceń/zarządzania (Command Window) jest podstawowym oknem, do którego będziemy wpisywać polecenia i odczytywać wyniki obliczeń. Powinno być stosunkowo duże, żeby można było odczytać jak najwięcej informacji.
2. Przestrzeń robocza - w oknie Workspace wyświetlone będą wszystkie dane (zmienne, struktury itd.) które aktualnie są przechowywane w pamięci programu.
3. Katalog pracy - (Current Folder) katalog w którym zapisywane są pliki programów i z których pobierane są dane (bez konieczności specjalnego podawania ścieżki do pliku).

Okna można dowolnie *przeciągać* i zmieniać wygląd interfejsu. Każde okno w pra-

wym górnym rogu ma własne *menu*, jest tam opcja Undock, która pozwala odłączyć dane okno od całego interfejsu. Opcja Dock jest działaniem odwrotnym.

W przypadku doprowadzenia interfejsu do takiego stanu, że nie jesteśmy w stanie odwrócić tych działań należy na górnym pasku w zakładce **Home** rozwinąć (a) **Layout** i wybrać opcję **Default** - zresetowanie widoku interfejsu.

Zaraz obok są opcje (b) **Preferences**, gdzie można dostosować program do swoich potrzeb (np. zmienić czcionkę i jej rozmiar, kolory, katalogi zapisu danych i wiele innych).



Rysunek 1: Interfejs użytkownika oprogramowania Matlab R2020a

2 Polecenia podstawowe

Wszystkie poniższe polecenia należy wpisywać w **Command Window** i obserwować rezultaty, również w tym oknie. Każdą linijkę należy potwierdzić klawiszem **Enter**. W większości przypadków wybrane liczby są przykładowe, a student może wprowadzać dowolnie inne.
Uwaga. Proszę w tym momencie NIE wpisywać kodu do okienka *Editor*, skorzystamy z niego dopiero w dalszej części zajęć.

1. **Zapisanie w pamięci liczby i przypisanie jej do zmiennej np. *a***

```
a=7
```

Do zmiennej o symbolu **a** została przypisana wartość **7** i zapisana w pamięci. Rezultat tego działania został wyświetlony na ekranie. Matlab rozróżnia małe i wielkie litery, stąd **a** i **A** to dwa różne symbole.

2. Zapisanie liczby ułamkowej w postaci dziesiętnej i rola średnika

```
b=5.7;
```

Do **b** przypisano wartość **5.7**. Część ułamkowa jest oddzielona **KROPKĄ!** Z kolei średnik postawiony na końcu linijki polecenia decyduje o tym, czy efekt wpisanego polecenia zostanie wyświetlony w oknie poleceń (porównaj z punktem 1). Ma to szczególne znaczenie przy bardziej rozbudowanych programach, gdzie wyświetlanie zbędnych informacji (np. z każdego kroku) może znacząco wydłużyć czas wykonywania skryptu.

3. Podstawowe działania arytmetyczne
mnożenie,
/ dzielenie,
+,- dodawanie, odejmowanie,
^ potęgowanie.
Wykonać kilka podstawowych działań np.

```
c=a*b
d=b^a
```

etc.

4. Przestrzeń robocza
Wszystkie zmienne i stałe są zapisywane oraz przechowywane w pamięci. W Matlabie tę pamięć nazywamy przestrzenią roboczą (**Workspace**). W każdej chwili można sprawdzić jakie dane zostały już zapisane poleceniem

```
who
```

z kolei, żeby uzyskać bardziej szczegółowe informacje (np. ilość zajmowanego w pamięci miejsca) należy użyć

```
whos
```

Na ekranie powinny zostać wyświetlone informacje dotyczące tych zmiennych, które zostały wcześniej zadeklarowane.

5. Usuwanie zmiennych z przestrzeni roboczej i czyszczenie konsoli
Można usuwać pojedyncze zmienne z pamięci, np.

```
clear a
```

Sprawdzić poleceniem **who**, czy zmienna została usunięta.
W celu usunięcia wszystkich zmiennych z pamięci, należy użyć polecenia

```
clear all
```

Przydatnym poleceniem jest również

```
clc
```

które służy do czyszczenie okna poleceń z niepotrzebnych już informacji.

Od polecenia **clear all** oraz **clc** będziemy zaczynali prawie każdy skrypt.

3 Tablice

6. Wprowadzanie tablic do pamięci
Do pamięci komputera należy wprowadzić tablicę A o wymiarze $2x2$, o dowolnych wyrazach

```
A=[1 4; 2 5]
```

Kwadratowy nawias [otwiera tablicę, a kolejne elementy wpisywane są wierszami. W pierwszym wierszu są elementy **1** i **4**, które mogą być oddzielone **SPACJĄ** lub **PRZECINKIEM**, przejście do kolejnego wiersza odbywa się poprzez użycie **ŚREDNIKA**. Macierz zamykamy].
Wprowadzić drugą, dowolną tablicę **B** również o wymiarze $2x2$.
Tablicę prostokątną (dwuwskaźnikową) możemy nazywać również macierzą.

7. Operacje macierzowe i tablicowe - WAŻNE
Na wprowadzonych tablicach/macierzach można wykonywać operacje **TABLICOWE** lub **MACIERZOWE**. Zrozumienie różnicy pomiędzy tymi operacjami jest kluczowe w kontekście zrozumienia wszystkich kolejnych zajęć z Metod Obliczeniowych.
Wyjaśnijmy to na przykładzie.

Do pamięci (przestrzeni roboczej) wprowadzono 2 tablice:

```
A=[1 4; 2 5]
B=[2 3; 2 3]
```

a) Mnożenie macierzowe
Zapiszmy w Command Window:

```
C=A*B
```

A i **B** zostają wymnożone przez siebie zgodnie z zasadą mnożenia macierzy. Czyli polecenie operacji arytmetycznej zapisujemy dokładnie tak jak dla operacji mnożenia pojedynczych liczb. **Operacje macierzowe** są więc ustawione jako *domyślne* w Matlabie.
Otrzymany wynik to:

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 10 & 15 \\ 14 & 21 \end{bmatrix}$$

czyli w macierzy **C** na przecięciu pierwszego wiersza i pierwszej kolumny mamy liczbę $1 \cdot 2 + 4 \cdot 2 = 10$, itd.

b)Mnożenie tablicowe
Żeby wykonać operację tablicową należy użyć OPERATORA TABLICOWEGO w postaci **KROPKI**, która jest stawiana przed operatorem matematycznym. Zapiszmy:

```
D=A.*B
```

Tablice **A** i **B** zostają wymnożone przez siebie tablicowo. Oznacza to, że wymnożone zostają przez siebie tylko wyrazy, znajdujące się na tych samych pozycjach.
Otrzymany wynik:

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 2 & 12 \\ 4 & 15 \end{bmatrix}$$

Tym razem w tablicy **D** na przecięciu pierwszego wiersza i pierwszej kolumny mamy liczbę $1 \cdot 2 = 2$, itd.
Porównaj wynik z mnożeniem macierzowym.

UWAGA:W operacji tablicowej **kropka** jest stawiana przed operatorem matematycznym i jest bezpośrednio z nim związana. Nie należy kojarzyć w tej operacji, że **kropka** jest stawiania za zmienną (zostanie o tym jeszcze wspomniane dalej).

c) Inne operacje tablicowe i macierzowe
Oczywiście operacje tablicowe i macierzowe nie dotyczą tylko mnożenia. Wykonaj podobne operacja dla potęgowania:

```
E=A^2
F=A.^2
```

Porównaj wyniki.
W pierwszym przypadku (potęgowanie macierzowe) wynik jest taki jak przy przemnożeniu przez siebie dwóch macierzy **A**.

W drugim przypadku (potęgowanie tablicowe) podnosimy do kwadratu każdy element w tablicy z osobna.

Przy dodawaniu lub odejmowaniu nie ma to większego znaczenia ponieważ wynik obu operacji będzie taki sam.

Operacje tablicowe są podstawowym elementem Metod Obliczeniowych (Metod Numerycznych) i bardzo często będziemy z nich korzystać.

8. Pozostałe tablice
W Matlabie jest wiele zaimplementowanych poleceń umożliwiających szybkie tworzenie tablic o dowolnym wymiarze. Poniżej wybrane z nich.

Wprowadzenie macierzy o wymiarze $6x6$ wypełnionej zerami:

```
e=zeros(6)
```

Macierz PROSTOKĄTNA wypełniona zerami **(pierwsza liczba określa liczbę wierszy, druga liczbę kolumn**):

```
f=zeros(4,8)
```

Macierz wypełniona jedynkami

```
g=ones(9)
```

Zadanie 1: Jak utworzyć macierz wypełnioną samymi siódemkami?

Macierz jednostkowa

```
h=eye(8)
```

Kwadrat magiczny

```
k=magic(5)
```

Macierz o elementach losowych z zakresu od **0** do **1**

```
m=rand(6)
```

Zadanie 2: Jak wprowadzić macierz z losowymi elementami z zakresu od **0** do **50**?

Zadanie 3: Jak wprowadzić macierz z losowymi elementami z zakresu od **-50** do **100**?

9. Transponowanie, odwracanie macierzy i zmienna ans
Podstawową operacją wykonywaną na macierzy jest transpozycja

```
n=m'
```

Pod zmienną **n** jest zapisana macierz będąca transpozycją macierzy **m**.
Druga operacja to odwrócenie macierzy poleceniem

```
inv(m)
```

Jeżeli polecenie zostało zapisane dokładnie tak jak w linijce powyżej, to macierz odwrotna do **m** została zapisana pod nową zmienną **ans** (answer). Ponieważ użytkownik sam nie zadeklarował zmiennej, pod którą ma zostać zapisany rezultat operacji to program sam stworzył zmienną **ans**, pod którą przechowuje wynik. Użyj polecenia **who**, żeby sprawdzić czy taka zmienna istnieje. Wpisz potem w konsoli

```
ans
```

żeby sprawdzić co jest przypisane do tej zmiennej.

UWAGA! Jest tylko jedna zmienna *ans*, pod którą zapisywany jest wynik ostatniej operacji, dla której nie zadeklarowano zmiennej (poprzedni wynik zostanie nadpisany). Dlatego też podczas pisania programów użytkownik nie operuje na **ans**, tylko sam definiuje zmienne.

10. Adresowanie elementów tablic/macierzy
Wprowadź tablicę o wymiarze $4x4$ np. poleceniem (możesz wprowadzić inną tablicę):

```
A=magic(4)
```

Wybierz pojedynczy element, leżący na przecięciu 2giego wiersza i 3ciej kolumny, i wyświetl go:

```
A(2,3)
```

Na PIERWSZYM miejscu jest numer WIERSZA, natomiast na drugim numer KOLUMNY!

Wybrany element oczywiście można przypisać do innej zmiennej

```
b=A(2,3)
```

inaczej zostanie zapisany pod zmienną *ans*.
Można też zastąpić wybrany element w tablicy inną liczbą np.

```
A(2,3)=50
```

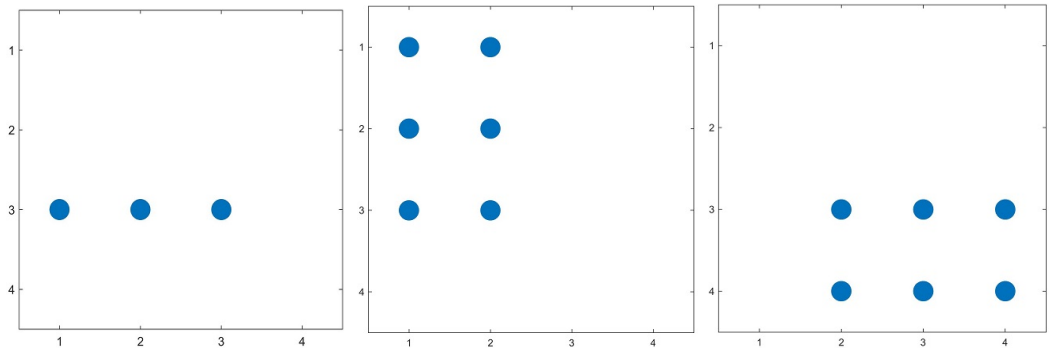
Wybranie wszystkich elementów z drugiego wiersza

```
A(2,:)
```

Symbol : oznacza wszystkie elementy ze zbioru. Czyli powyższy zapis należy rozumieć jako wybór elementów leżących na przecięciu 2giego wiersza i wszystkich kolumn.
Analogicznie wybór wszystkich elementów z 4tej kolumny

```
A(:,4)
```

Elementy można wybierać w bardziej złożony sposób. Żeby wybrać elementy zaznaczone na Rys. 2a należy zapisać



Rysunek 2: Wybór elementów. Przykład a), b) i c).

```
A(3,1:3)
```

gdzie zapis $1:3$ można w uproszczeniu rozumieć jako wybór elementów od 1 do 3 kolumny (omówimy to dokładniej w dalszej części).

Wybór elementów według przykładu przedstawionego na środku to

```
A(1:3,1:2)
```

Zadanie 4: Wybierz elementy z macierzy **A** według zaznaczenia na rysunku po prawej.

Rozwiązania do wszystkich zadań znajdziesz w rozdziale 9, jednak w pierwszej kolejności postaraj się samodzielnie je rozwiązać. Są krótkie i proste.

4 Wektory

Rozdział ten jest prawdopodobnie najważniejszy w kontekście metod numerycznych (obliczeniowych). Wektory w metodach numerycznych oraz samym Matlabie pojawiając się na każdym kroku, dlatego też ich zrozumienie jest kluczowe.

Co to jest **wektor**?
Pod pojęciem wektora będziemy rozumieli tablicę jednowymiarową (jednowskaźnikową), czyli np.:

$$\begin{bmatrix} -7 & 2 & 4 & -2 & 10 & 0 & 4 \end{bmatrix}$$

Ogólnie wektor możemy przedstawić w postaci matematycznej:

$$x = \begin{bmatrix} x_1 & x_2 & \dots & x_k & \dots & x_{n-1} & x_n \end{bmatrix}$$

Gdzie indeks dolny oznacza indeks/adres elementu w wektorze **x**. Symbol **n** oznacza całkowitą liczbę w wektorze **x**. Symbol **k** odnosi się do adresu elementu, który może zmieniać się w zakresie od **1** do **n**, czasami mówimy k-ty (czyt. katy) element w wektorze **x**.

Czasami w książkach lub na wykładzie można spotkać się z zapisem:

$$x = \left[\begin{array}{cccccccc} x_0 & x_2 & \dots & x_i & \dots & x_{n-2} & x_{n-1} \end{array} \right]$$

Zazwyczaj indeks wyrazu opisany jest symbolem **i**. Ponieważ w Matlabie do symboli **i** oraz **j** przypisana jest wartość urojona, dlatego dobrze wypracować sobie nawyk korzystania z innych symboli niż te dwa. Ja na zajęciach będę używał głównie symbolu **k**. Z matematycznego punktu widzenia to oczywiście nie ma znaczenia, byle zachować konsekwencję.

Drugą kwestią jest to, że pierwszy element w wektorze ma adres **0**, a ostatni **n-1**, przy czym liczba elementów w wektorze nadal wynosi **n**. Wynika to z tego, że adresowanie w niektórych programach (np. język C) jest od **zera**. I łatwiej przenieść taki zapis matematyczny bezpośrednio do programu.

W Matlabie (jak już można było zauważyć przy tablicach) indeksowanie jest od **1**, dlatego też ja będę stosował ten pierwszy zapis. Przy czym należy zaznaczyć, że oba zapisy oznaczają w zasadzie to samo i bardzo łatwo można się pomiędzy nimi *przełączać* dodając lub odejmując wartość 1 od wszystkich indeksów.

Wyrazy w wektorze mogą być całkowicie dowolne, jednak w metodach obliczeniowych (numerycznych) najczęściej będą tworzyły ciąg arytmetyczny, czyli na przykład:

$$x_2 = x_1 + h$$

$$x_3 = x_2 + h = x_1 + 2 \cdot h$$

$$x_8 = x_1 + 7 \cdot h$$

lub bardziej ogólnie

$$x_k = x_1 + (k-1) \cdot h$$

gdzie **h** jest różnicą pomiędzy kolejnymi elementami wektora. W matematyce tę różnicę najczęściej oznacza się symbolem **r**, jednak my będziemy oznaczać ją symbolem **h** i nazywać **KROKIEM** (czasami też skokiem).

Przykład:

$$\left[\begin{array}{cccccccc} -4 & -2 & 0 & 2 & 4 & 6 & 8 & 10 \end{array} \right]$$

jest to wektor **-4** do **10** z krokiem **2**.

11. Wprowadzanie wektorów
Utworzyć wektor o elementach od **5** do **15**, z różnicą pomiędzy kolejnymi elementami (krokiem) **h=1**

```
x=5:15
```

Utworzyć wektor o elementach od **5** do **10** i różnicą pomiędzy kolejnymi elementami równą **h=0.2**

```
x=5:0.2:10
```

Jak widać, domyślny krok **h** jest równy **1**, jeżeli chcemy aby ta wartość była inna należy ją podać.
Utworzyć wektor o dowolnych elementach

```
y=[2 5 17 19 114]
```

Wektor jest wprowadzany tak jak tablica, z tą różnicą że wpisujemy tylko pojedynczy wiersz.

12. Operacje na wektorach
Elementy wektora wybieramy podobnie jak w przypadku tablic prostokątnych z tą różnicą, że podawany jest tylko jeden indeks (wektor ma jeden wymiar). Wybór 4tego elementu z wektora **x**

```
x(4)
```

lub wybór i przypisanie go do jakiejś innej zmiennej

```
c=x(4)
```

Nadpisanie elementu inną liczbą

```
x(4)=12
```

Usunięcie 4tego elementu z wektora **x**

```
x(4)=[]
```

oznacza to że w miejsce 4tego elementu zostaje wstawiony pusty wektor (czyli NIC), cały wektor **x** zostaje skrócony o jeden element.
Zaadresowanie ostatniego elementu

```
x(end)
```

Dopisanie liczby **17** na końcu wektora

```
x(end+1)=17
```

Sprawdzenie **DŁUGOŚCI** wektora, czyli sprawdzenie ile elementów znajduje się w wektorze (bardzo często używane polecenie)

```
k=length(x)
```

do zmiennej **k** przypisano liczbę równą długości wektora **x**.
(Często popełniany błąd w tym poleceniu to pisanie **lenght**, należy zwrócić uwagę, że na końcu wyrazu **t** jest przed **h** -» **lengTH**)

Wybranie zakresu elementów z wektora **x**, np. od **4** do **9**

```
x(4:8)
```

Przy omawianiu tablic była informacja, że w uproszczeniu można to interpretować jako wybranie elementów od 4 do 8. W rzeczywistości podany w nawiasie zakres **4:8** to nic innego jak wektor. Czyli wybieramy z wektora **x** wyrazy za pomocą innego wektora! Równie dobrze moglibyśmy zapisać to w postaci:

```
x([4 5 6 7 8])
```

Oznacza to, że z wektora możemy wybierać dowolnie różne wyrazy. Na przykład:

```
x([2 5 11])
```

czyli wybieramy wyrazy o indeksach 2, 5 i 11.
Możemy utworzyć nowy wektor, który będzie zawierał co drugi element z wektora **x**

```
d=x(1:2:end)
```

Możliwości jest wiele, a zrozumienie tego będzie bardzo pomocne w przyswojeniu wiedzy na kolejnych zajęć.

13. Łączenie wektorów
Wektory można dowolnie modyfikować i łączyć ze sobą. Można połączyć wektor **x** z wektorem **y**, tworząc nowy wektor **z**

```
z=[x y]
```

Można też wektory uzupełnić dowolnymi liczbami i łączyć z fragmentami innych wektorów np.

```
zz=[3 11 21 x(3:8) 4:8 12 y(1:5)]
```

Uwaga. Na pierwszy rzut oka wektory wydają się proste oraz zrozumiałe i tak w rzeczywistości jest. Jednak najczęstszym *grzechem* studentów podczas praktycznych zajęć jest mylenie indeksu/adresu elementu z wartością, która pod nim stoi. Przykład takiego błędu zostanie opisany w kolejnym rozdziale.

5 Najczęstsze błędy w Matlabie

W przypadku wprowadzenia niepoprawnego kodu do Matlaba, w Command Window zostanie wyświetlony na czerwono komunikat opisujący błąd. Program podaje też w której linijce popełniono błąd. Proszę czytać czytać komunikaty i poprawiać błędy. Poniżej przedstawiono najpopularniejsze komunikaty błędów.
Teraz możesz pominąć ten rozdział i wrócić w razie potrzeby.

1.

Adres do elementu w tablicy (wektorze) przekroczył rozmiar tablicy (wektora). Przykładowo wektor zawiera 10 elementów, a chcemy zaadresować element 11 (którego nie ma).

```
Index exceeds the number of array elements (10).

Error in bledy (line 6)
    x(k)

Array indices must be positive integers or logical values.

Error in bledy (line 6)
    x(k)
```

2.

Adres do elementu w tablicy (wektorze) musi być liczbą naturalną. Taki błąd wystąpi np. w przypadku gdy zaadresujemy element w wektorze `x(5.5)`. Adresowanie zaczynamy od 1, więc `x(0)` również jest błędne.

3.

```
Error: File: bledy.m Line: 4 Column: 12
Invalid expression. When calling a function or indexing a variable, use
parentheses. Otherwise, check for mismatched delimiters.
```

Komunikat ten może wyświetlić się w kilku przypadkach, najczęściej jednak informuje o nieprawidłowej liczbie nawiasów. Np. brak domknięcia wyrażenia matematycznego: `x=(2*x*(4+9*y)`

4.

```
Error: File: bledy.m Line: 4 Column: 4
Invalid expression. Check for missing multiplication operator, missing or
unbalanced delimiters, or other syntax error. To construct matrices, use
brackets instead of parentheses.
```

Niepoprawnie zapisane wyrażenie. Np. pominięcie znaku mnożenia przy zapisie funkcji `y=2x`, lub zastosowanie okrągłych nawiasów zamiast kwadratowych przy wprowadzaniu macierzy.

5.

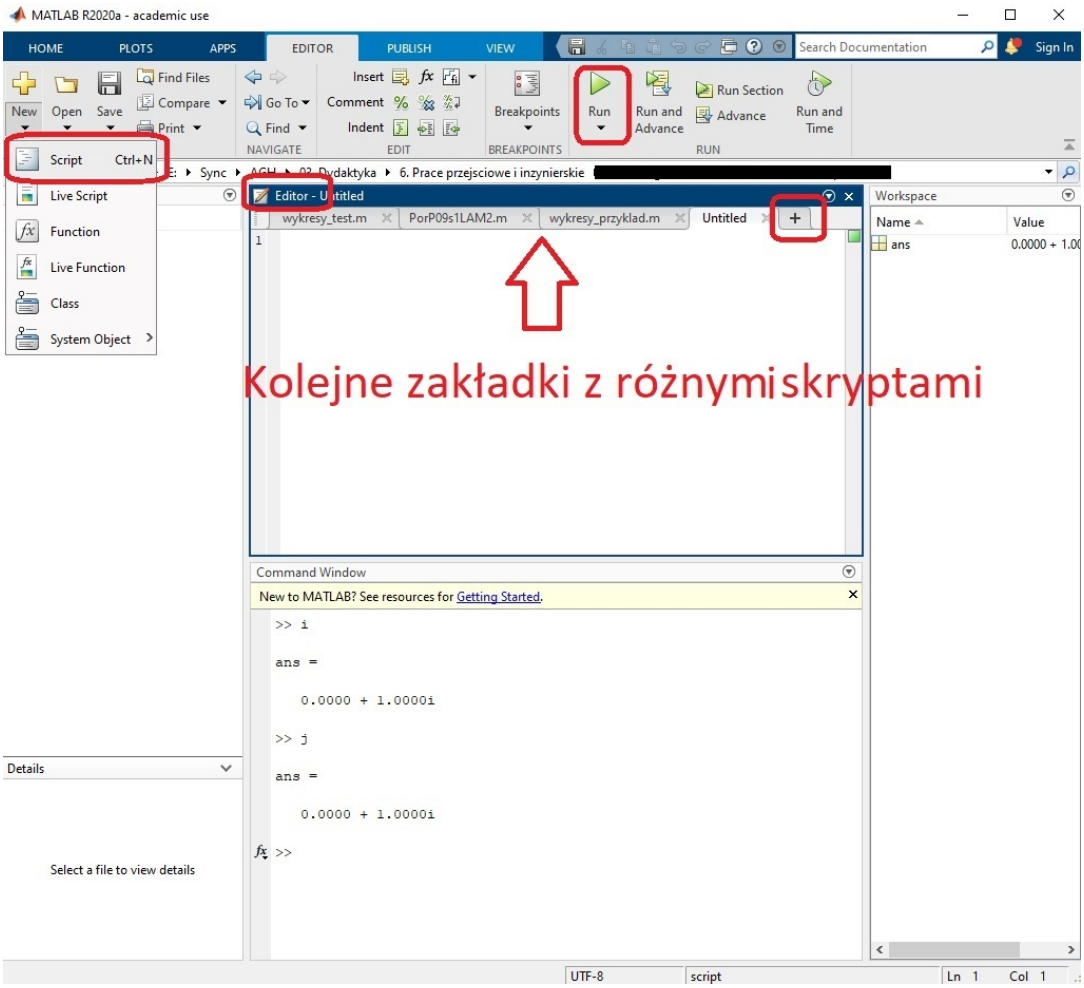
```
Error using plot
Vectors must be the same length.

Error in bledy (line 7)
plot(x,y)
```

Żeby narysować wykres poleceniem **plot** oba wektory muszą mieć taką samą liczbą elementów. Długość wektorów można sprawdzić poleceniem **whos** lub w **Workspace**.

6 Pierwszy program i zastosowanie wektorów

14. M-plik

M-plik czyt. *emplik* (**M-file**) jest podstawowym plikiem Matlaba. Można go utworzyć wybierając z górnego menu **New -» Script**, lub za pomocą skrótu klawiszowego **Ctrl+N** (może się to nieznacznie różnić w zależności od wersji Matlaba). Otworzy się okno **Editor**, gdzie zapiszemy program. M-plik pozwala napisać kilka linijek skryptu, po czym uruchomić go w całości oraz oczywiście zapisywać. Późniejsze wprowadzanie poprawek nie wymaga wpisywania wszystkiego od nowa.



Rysunek 3: Uruchomienia okna edytora

Uruchomienie programu następuje poprzez wciśnięcie klawisza **F5**, bądź poprzez kliknięcie na zielony smybol trójkąta (Run) na głównym pasku w zakładce Editor. Istnieje kilka ważnych zasad jak NIE nazywać swoich programów.

Zasady nazywania plików:
1. Nazwa m-pliku **NIE może** zaczynać się od cyfry. Nie wolno nazywać programu 2lab, jednak można lab2.

2. **NIE stosować** SPACJI oraz KROPEK, unikać znaków # % $ etc.
3. **NIE nazywać** programów tak jak już są nazwane funkcje w Matlabie (np. length, for, while, plot), ponieważ spowoduje to tymczasowe nadpisanie danej funkcji i uniemożliwi korzystanie z niej.

15. Wykres funkcji - BARDZO WAŻNE
Zadaniem jest narysowanie wykresu funkcji $y = x^2 - 4x + 8$ w dziedzinie $x \in < -3, 3 >$, dla kroku $h = 1$.

Żeby zrealizować zadanie, należy wykonać 3 podstawowe czynności:
1. Utworzyć wektor **x** według treści zdania.
2. Dla każdego elementu w wektorze **x** trzeba obliczyć wartość funkcji **y** według podanej zależności.
3. Narysować wykres, łącząc linią obliczone punkty.

Zanim przejdziemy do bardziej szczegółowych objaśnień narysujmy wykres funkcji **y**.
Uwaga! Przepisuj programy samodzielnie, zamiast je przeklejać, dzięki temu nauka będzie efektywniejsza.
Program (zapisz całość w skrypcie (okno Editor) i uruchom):

```
clear all
clc
close all
x=-3:3
y=x.^2-4*x+8
plot(x,y,'*-')
```

Otrzymany wykres:
Objaśnienie:
1. Polecenia **clear all** i **clc** usuwają wszystkie dane z przestrzeni roboczej i czyszczą konsolę, polecenie **close all** zamyka wszystkie okna z wykresami z poprzednich uruchomień programu. Pozwala to pozbyć się wszystkich niepotrzebnych danych z innych programów i od tych poleceń będziemy zaczynali każdy program.

2. Dwie kolejne linijki to utworzenie wektora **x** oraz **y**.
Wpisz w konsoli (Command Window):

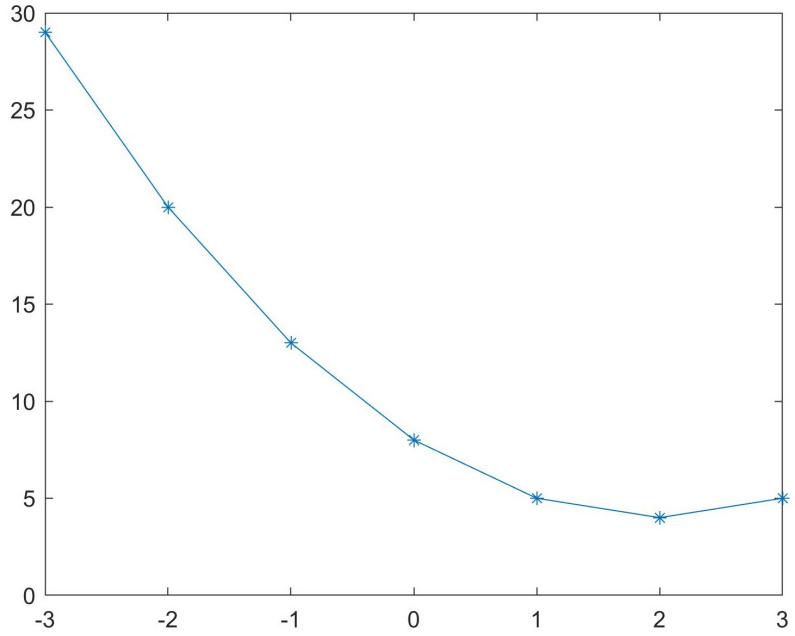
```
x
```

```
y
```

lub

```
disp(x)
```

```
disp(y)
```



Rysunek 4: Wykres funkcji

lub kliknij 2 razy na te zmienne w okienku Workpace. Zostaną wyświetlone dwa następujące wektory:

$$x \quad \begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}$$

$$y \quad \begin{bmatrix} 29 & 20 & 13 & 8 & 5 & 4 & 5 \end{bmatrix}$$

Dla pierwszego argumentu w wektorze **x** czyli $x_1 = -3$, została obliczona wartość funkcji $y_1 = 29$ (podstawiając do wzoru funkcji za -3 otrzymamy taką wartość). Dla argumentu $x_2 = -2$ wartość funkcji jest równa $y_2 = 20$, itd. Liczby te parami stanowią współrzędne punktów, czyli:

$(-3, 29)$
$(-2, 20)$
...
$(3, 5)$

W funkcji **y** potęgowanie przy jednomianie x^2 zostało zapisane z KROPKĄ, ponieważ wykonujemy tutaj operacje TABLICOWE. Czyli na każdym elemencie z osobną. Do wzoru na funkcję podstawiany jest pierwszy elementy z wektora **x** (-3) i obliczana jest dla niego pierwsza wartość w wektorze **y** (29), itd. Dalej przy operacji mnożenia $4x$ nie ma kropki przed *, ponieważ przy mnożeniu przez stałą wartość wynik operacji macierzowej i tablicowej będzie taki sam. Wcześniej było informacja, że KROPKA jest związana ze znakiem arytmetycznym a nie zmienną, więc w jednomianie $4x$ nie ma też kropki za x (co jest często popełnianym błędem).
Oczywiście najpierw definiujemy **x** a dopiero potem **y**, ponieważ program nie może wykonać obliczeń, na podstawie zmiennych których jeszcze nie ma podanych.

Zapamiętaj, że przy zapisywaniu funkcji w przedstawiony sposób, wykonujemy operacje TABLICOWE!!!

3. Polecenie **plot** otwiera nowe okno z wykresem. Należy pamiętać, że najpierw podajemy wektor osi odciętych (oś pozioma - ogólnie x), a potem wektor osi rzędnych (oś pionowa - ogólnie y). Oczywiście na osiach mogą znaleźć się inne symbole, np. przedstawiając zależność napięcia od czasu będzie to *plot(t,U)*.
Mówiąc ściślej, polecenie **plot** nanosi na wykres punkty o wyliczonych współrzędnych - te które opisano powyżej, czyli pierwszy punkt o współrzędnych (-3,29) itd. (na wykresie punkty te oznaczone są gwiazdką * są to tzw. markery lub znaczniki). Natomiast punkty łączone są linią prostą.

W poleceniu **plot**, na końcu zapisano **'*-'**, co decyduje o sposobie wyświetlenia wykresu (markery/znaczniki w postaci gwiazdek i wykres narysowany linią ciągłą). Jeżeli usuniesz ten fragment (do postaci **plot(x,y)**), pozbędziesz się markerów.

UWAGA! Najczęstszy błąd.
BARDZO często popełniany przez Studentów błąd jest powiązany z tym o czym już wcześniej wspominałem - Mylenie indeksu/adresu elementu z wartością, która pod tym adresem stoi.
Przykład: Polecenie brzmi: Proszę wyświetlić wartość funkcji dla argumentu $x = 2$.
Niepoprawna odpowiedź jaka często pada:

```
y(2)
```

Odpowiedź ta jest związana z przyzwyczajeniem z matematyki, gdzie powszechny jest zapis $y(x)$ lub $f(x)$.
Odwołajmy się teraz do wyznaczonych wektorów:

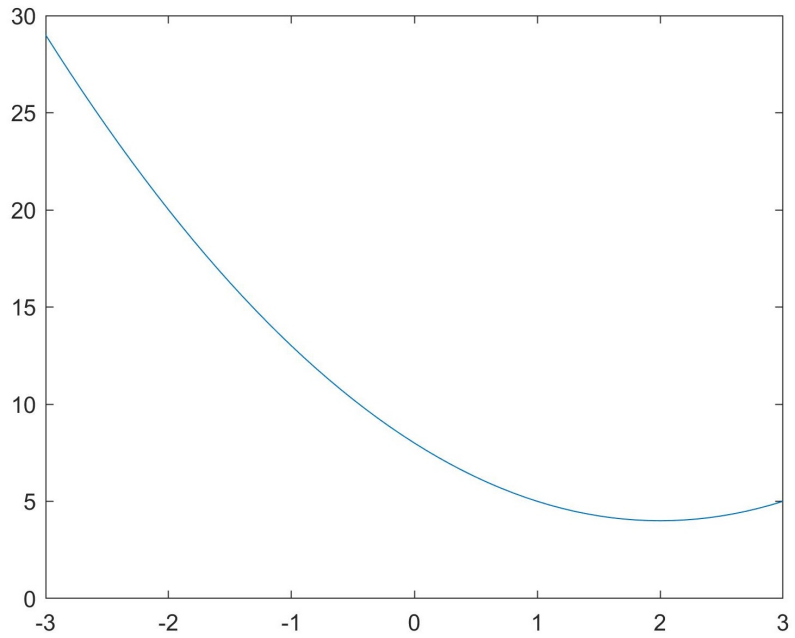
$$x \quad \begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}$$

$$y \quad \begin{bmatrix} 29 & 20 & 13 & 8 & 5 & 4 & 5 \end{bmatrix}$$

Jeżeli Student zapisze
`y(2)` lub `disp(y(2))`
to wyświetlona zostanie wartość 20, czyli drugi element w wektorze **y**, a tym samym wartość funkcji dla argumentu $x = -2$, a nie 2 tak jak w treści zdania.

Żeby poprawnie udzielić odpowiedz trzeba określić pod jakim indeksem w wektorze **x** stoi wartość 2 (czyli tutaj $x_6 = 2$), a następnie tym indeksem zaadresować element w wektorze **y**. Poprawna odpowiedź to:
`y(6)` lub `disp(y(6))`
czyli wynikiem jest wartość 4.
W takiej sytuacji oba wektory będą miały taką samą długość, więc adresy do współrzędnych tego samego punktu będą takie same.

Zadanie 5: Jak już wcześniej powiedziano, obliczone punkty są na wykresie zaznaczone markerami i połączone linią prostą. Jednak wykres ten nie wygląda zbyt *ładnie* (wyraźnie widać załamania linii). Co należy zrobić, żeby *wygładzić* wykres funkcji?
Zmodyfikuj program aby uzyskać mniej więcej taki sam wykres jak na rys. 5.
Uwaga. Nawet jeżeli udało Ci się samodzielnie rozwiązać to zadanie, to i tak zajrzyj do odpowiedzi na końcu, ponieważ jest tam wyjaśniona jeszcze jedna bardzo ważna kwestia.



Rysunek 5: Poprawiony wykres funkcji

Ponieważ zrozumienie tego punktu 15, jest bardzo ważne najlepiej przeczytaj wyjaśnienia raz jeszcze np. jutro.
Narysuj samodzielnie ze 2 - 3 wykresy innych funkcji.

16. Modyfikacja wykresu funkcji

W tym punkcie nauczymy się opisywać wykresy oraz zmieniać ich wygląd, co pozwoli Ci wykorzystać Matlaba do sporządzania wykresów w sprawozdaniach (nie tylko z przedmiotu MOiPE). Wiedza ta, przyda się również do realizacji zajęć Projektowych z MOiPE.

W oknie poleceń Command Window (nie zamykaj okna edytora z programem z podpunktu 15) należy wpisać polecenie

```
help plot
```

zostanie wyświetlona pomoc do polecenia **plot**. Tak samo można uzyskać pomoc do

innych funkcji np. **help length, help for,** etc.

Tabela 1: Parametry funkcji *plot*

Kolor	Znacznik	Rozdzaj linii
y	o	-
m	+	–
c	*	:
r	.	-.
g	x	(none)
b	s	
w	d	
k	v	
	>	
	<	
	p	
	h	

Należy przewinąć ekran pomocy do miejsca gdzie będą 3 kolumny (w Tab 1 przedstawiono te 3 kolumny, gdyż w niektórych wersjach Matlaba, nie wyświetlają się bezpośrednio w pomocy). Każda z kolumn odpowiada za inny parametr wykresu:
1. Pierwsza kolumna decyduje o kolorze wykresu (domyślnie niebieski).
2. Druga to symbole, które pojawiają się na wykresie w punktach o współrzędnych zapisanych w wektorach x i y czyli znaczniki/markery (domyślnie brak). Punkty te na wykresie łączone są linią prostą.
3. Trzecia kolumna decyduje o rodzaju linii, jaką narysowany jest wykres np. ciągła, kreskowana, brak linii (domyślnie ciągła).

Zadanie: Wybrać po jednym dowolnym symbolu z każdej kolumny i zmodyfikować polecenie **plot** w swoim programie zgodnie z przykładem zapisanym w pierwszej linijce pod kolumnami np. `plot(x,y,'ro:')`, ponownie uruchomić program. Sprawdzić różne warianty.
Uwaga: zawartość pomocy może się różnić w zależności od wersji Matlaba.

Inne przydatne funkcje pozwalające opisać wykres to (polecenia należy dopisywać do programu (w skrypcie) pod poleceniem **plot**:
Nałożenie siatki

```
grid on
```

lub bardziej zagęszczonej

```
grid minor
```

Dodanie tytułu do wykresu

```
title('Prędkość punktu w funkcji czasu')
```

Opisanie osi odciętych

```
xlabel('Czas t [s]')
```

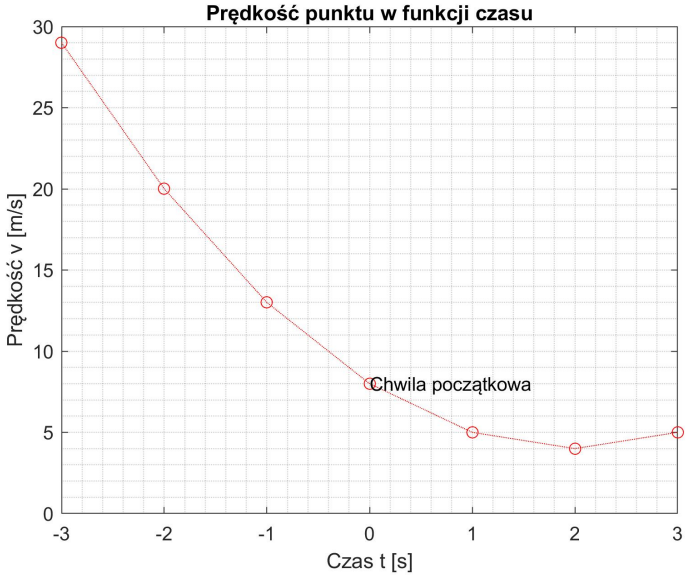
Opisanie osi rzędnych

```
ylabel('Prędkość v [m/s]')
```

Dodanie opisu na wykresie

```
text(0,8,'Chwila początkowa')
```

gdzie 0 i 8 to współrzędne punktu, w którym pojawi się opis.



Rysunek 6: Opisany wykres funkcji

Są to tylko podstawowe polecenia opisujące wykres, ponieważ zmieniać na wykresie można w zasadzie wszystko (wielkość znaczników, zakres osi, czcionkę i jej rozmiar i wiele innych). Więcej informacji można znaleźć w Internecie, w pomocy Matlaba i na stronie (jest tam również forum) `https://www.mathworks.com/`.

7 Pętle i instrukcja warunkowa

Podstawowymi strukturami każdego języka programowania są **Pętle** oraz **Instrukcja warunkowa**. Ich działanie jest dokładnie takie samo jak w innych językach (np. C), jednak składnia może się różnić. Opiszemy te struktury w Matlabie na bardzo prostych przykładach.

17. Pętla FOR
Pętlę for stosujemy wówczas kiedy jeszcze przed rozpoczęciem pętli, jesteśmy w stanie określić ile razy zostanie ona wykonana.

Komentarze i opis do skryptów można dodawać z %. Można również zaznaczać fragmenty programu i skrótem **Ctrl+R** oraz **Ctrl+T** wyłączać je i włączać.

Treść zadania: Za pomocą pętli FOR utworzyć wektor złożony z 10 elementów, w którym kolejne wartości będą równie 50 minus adres pod którym stoją.

Czyli chcemy otrzymać takie rozwiązanie:

$$x \quad \begin{bmatrix} 49 & 48 & 47 & 46 & 45 & 44 & 43 & 42 & 41 & 40 \end{bmatrix}$$

Pierwszy element stoi pod adresem 1, więc wyraz ten jest równy 50-1=49. Zapis matematyczny, dla pierwszego elementu to:

$$x_1 = 50 - 1$$

dla drugiego elementu będzie to:

$$x_2 = 50 - 2$$

a dla ostatniego

$$x_{10} = 50 - 10$$

Żeby wprowadzić do wektora **x** pierwszy element, to w Matlabie moglibyśmy zapisać (nie wpisuj tego na razie)

```
x(1)=50-1
```

i tak z każdym kolejnym wyrazem.
Jednak my chcemy to zautomatyzować, ponieważ dla np. 100 000 wyrazów nikt ręcznie, wyraz po wyrazie, nie będzie tego wprowadzał.

Wszystkie powyższe zapisy matematyczne, możemy zapisać w bardziej ogólny i zwięzły sposób, czyli:

$$x_k = 50 - k$$

gdzie **k** jet numerem indeksu i zmienia się od **1** do **10**. Odpowiada to treści naszego zadania (przeczytaj ją raz jeszcze).
Musimy to tylko wprowadzić do programu, czyli zrobić tak żeby indeks **k** zmieniał się sam w podanym zakresie i do tego posłuży właśnie pętla **for**.

Zapisz skrypt (w oknie Editor) i go uruchom:

```
clear all
clc
close all

for k=1:10
  x(k)=50-k
end
```

Właściwy program składa się z 3 linijek:
1. Warunek pętli `for k=1:10`, czyli w każdym kroku zmienna **k** przyjmuje wartość od 1 do 10. Zwróć uwagę, że `1:10` to nic innego jak WEKTOR (tak znowu pojawiają się wektory). Więc może on przyjmować dowolne wartości.
Można by np. zapisać `for k=[3 5 9]`, pętla wtedy zostanie wykonana 3 razy, a **k** przyjmie kolejno wartości 3, 5 i 9.
2. W pętli zapisaliśmy w języku Matlaba to, co chwilę wcześniej określiliśmy zapisem matematycznym. Ponieważ **k** w każdym kroku pętli będzie się zmieniało od 1 do 10, to do wektora będą wstawiane kolejne wyrazy. W pierwszym x(1)=50-1, itd. Jeżeli na końcu tej linijki nie dałeś/dałaś średnika, to w Command Window wyświetli się taka *choinka*. Każda linijka to jeden krok programu. Czyli np. po dwóch krokach w wektorze x mamy 2 elementy [49 48].
3. Polecenie **end** zamyka każdą strukturę (jest odpowiednikiem klamry } z innych języków programowania).

18. Pętla WHILE

Pętlę **while** stosujemy wtedy kiedy na początku wykonywania pętli nie wiemy jeszcze ile razy zostanie ona wykonana. **Pętla jest wykonywana dotąd, dopóki spełniony jest postawiony warunek.**

Treść zadania: Wylosuj liczbę wymierną z przedziału od 0 do 1000 i przerwij losowanie w momencie wylosowania liczby z przedziału od 999 do 1000.

Uwaga. Zanim przejdziesz dalej musisz wiedzieć, że przy źle postawionym warunku pętla **while** może wykonywać się w nieskończoność (czyli warunek zawsze będzie spełniony). Wówczas trzeba przerwać działanie programu (gdy w lewym dolnym rogu Matlaba zbyt długo będzie wyświetlał się komunikat **Busy**).
Zaznacz wtedy kursorem myszy okno Command Window i wciśnij **Ctrl+C**

Tutaj sprawa jest dosyć prosta, zapiszmy program a potem go omówimy:

```
clear all
clc
close all

a=0
while a<=999
  a=1000*rand(1)
end
```

Uruchom program.
1. Zacznijmy od warunku pętli. Warunek ten mówi, że dopóki **a** jest mniejsze lub równe 999, tak długo pętla będzie wykonywana i będą losowane nowe liczby. Dopiero wylosowanie liczby z tego wąskiego przedziału 999 – 1000 sprawi, że warunek nie będzie spełniony, a wykonywanie pętli zostanie zakończone.
2. Jest to fragment kodu losujący liczbę z przedziału od 0 do 1000. Wylosowana liczba zostaje przypisana do zmiennej **a**, poprzez nadpisanie liczby wylosowanej w kroku poprzednim (losowanie liczb z różnych zakresów jest opisane w odpowiedziach do Zadań 2 i 3).
3. Ponownie **end** zamyka strukturę.
4. Bardzo ważna jest linijka `a=0`, która pozwala zadeklarować zmienną **a**. Przypisujemy jej dowolną wartość spełniającą warunek pętli, po to żeby zainicjować pętlę. Jeżeli nie zapiszemy tej linijki (spróbuj) to program dojdzie do linijki **while** i wyświetli komunikat, że nie wie co to jest **a** i nie jest w stanie sprawdzić warunku. Niestety w Matlabie nie ma odpowiednika pętli `do while`, czyli takiej gdzie najpierw ją wykonujemy a dopiero na końcu sprawdzamy warunek.

Zadanie 6: Dodaj do pętli licznik, który poda ile razy odbyło się losowanie liczby.

19. Instrukcja warunkowa IF
W instrukcji IF stawiamy warunek, jeśli jest on prawdziwy to wówczas wykonywana jest operacja bezpośrednio pod warunkiem, jeśli nie to albo program nic nie robi, albo wykonuje inną operację, albo też sprawdzany jest kolejny warunek.

Zadanie 7: Napisać program, który będzie realizował następujące zadanie. Wygeneruje wektor **x** zawierający **20** losowych elementów, każdy przyjmujący wartość od 0 do 1. Należy sprawdzić każdy element wektora (np. za pomocą pętli FOR) i w zależności od tego w jakim przedziale znajduje się jego wartość zastąpić go elementem o innej wartości:
1. Jeżeli $x_k \leqslant 0.3$ to $x_k = 0$
2. Jeżeli $x_k > 0.8$ to $x_k = 1$
3. Jeżeli $0.3 < x_k \leqslant 0.8$ (czyli jedyna pozostała możliwość) to $x_k = x_k$ (pozostaje bez zmian).
Przykładowo jeśli element $x_k = 0.2379$ to znaczy że spełnia warunek 1. i należy go zastąpić liczbą 0.
Na końcu porównać oba wektory.

Dla ułatwienia przygotowano szkic programu, który należy uzupełnić:

```
clear all
clc
x=.....
for k=1:.....
    if x(k)......
       x(k)=0;
    elseif .......
       x(k)......
```

```
    else
       .....??????
    end
end
disp(x)
```

Rozwiązanie i objaśnienie w Odpowiedziach do zadań.

Podsumowanie instrukcji. W instrukcji przedstawiono tylko niezbędne informacje, które umożliwią realizację kolejnych zajęć z przedmiotu Metody Numeryczne. Szczególnie ważne jest zrozumienie **wektorów** oraz informacji przedstawionych w punkcie 15.

Na kolejnych zajęciach odbędzie się krótka kartkówka (5 min), na której Student otrzyma do zapisania pojedyncze polecenia.
Przykładowa treść zdania: Utwórz wektor **x** od -4 do 6 z krokiem 0.5 i sprawdź jego długość.
Odpowiedź:

```
x=-4:0.5:6
k=length(x)
```

Takich zadań będzie 4 i będą do zapisania na kartce (dlatego będą właśnie takie krótkie).

8 Zadania do samodzielnego rozwiązania

Są to zadania opcjonalne, dla osób chcących przećwiczyć nabytą wiedzę. W razie pytań służę pomocą.

8.1 Tabliczka mnożenia

Wykorzystując pętle FOR, stworzyć tablicę o wymiarach $10x10$. Wyrazy tablicy mają utworzyć tabliczkę mnożenia.

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Rysunek 7: Tabliczka mnożenia

8.2 Tablica z uporządkowanymi elementami

Utworzyć tablicę o wymiarach $10x10$. Elementy mają być uporządkowane od 1 do 100 (pierwszy wiersz od 1 do 10, drugi od 11 do 20, itd.)

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Rysunek 8: Tablica z uporządkowanymi elementami

8.3 Szachownica

Utworzyć tablicę o wymiarach $10x10$. Wyrazy uzupełnić na przemian 0 i 1.

1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1

Rysunek 9: Szachownica

9 Odpowiedzi do zadań

Zadanie 1: Jak utworzyć macierz wypełnioną samymi siódemkami?

```
M=7*ones(9,5)
```

W nawiasie podajemy wymiar macierzy wypełnionej jedynkami i przemnażamy ją przez dowolną liczbę.

Zadanie 2: Jak wprowadzić macierz z losowymi elementami z zakresu od **0** do **50**?

Zasada działania jest taka sama jak w prawie każdym innym języku programowania. Polecenie losujące liczbę domyślnie robi to w zakresie od 0 do 1. Więc jeżeli chcemy powiększyć ten zakres, wystarczy przemnożyć tę losową liczbę, przez maksymalną wartość jaką chcemy uzyskać. Dla pojedynczej liczby będzie to:

```
k=50*rand(1)
```

(1) oznacza tutaj jednoelementową tablice.
Więc jeżeli chcemy otrzymać całą macierz losowych wartości od 0 do 50, musimy podać jej wymiary, np.:

```
k=50*rand(6,8)
```

Tutaj uwaga. Częstym błędem jest wpisywanie do nawiasów polecenia **rand** zakresu losowanej wartości, czyli np. rand(0,50) – **ŹLE**.

Zadanie 3: Jak wprowadzić macierz z losowymi elementami z zakresu od **-50** do **100**?

Rozwiązaniem zadania jest niewielka modyfikacja poprzedniego zadania. Po pierwsze obliczamy zakres pomiędzy minimalną i maksymalną wartością, który w tym przypadku wynosi **150**.
Zapisując

```
k=150*rand(6,8)
```

wylosujemy oczywiście liczby z zakresu od 0 do 150. Musimy więc cały ten zakres przesunąć w dół o 50. Czyli:

```
k=150*rand(6,8)-50
```

Wartość przy mnożeniu służy do opisania różnicy pomiędzy wartościam max i min, a odejmowana (lub dodawana liczba) przesuwa ten zakres w dół lub w górę. Liczba ta będzie pokrywała się wartością minimalną tego zakresu.

Zadanie 4: Wybierz elementy z macierzy **A** według zaznaczenia na rysunku po prawej.

```
A(3:4,2:4)
```

Zadanie 5: Jak już wcześniej powiedziano, obliczone punkty są na wykresie zaznaczone markerami i połączone linią prostą. Jednak wykres ten nie wygląda zbyt *ładnie* (wyraźnie widać załamania linii). Co należy zrobić, żeby *wygładzić* wykres funkcji?

Udzielnie odpowiedzi na samo pytanie jest proste. Jednak w tym pytaniu kryje się ważne zagadnienie. Najważniejszą rzeczą w metodach numerycznych/obliczeniowych jest to, że dysponujemy rozwiązaniem tylko w konkretnych punktach. Przytoczmy znowu wektory z naszego zadania:

$$x \quad \begin{bmatrix} -3 & -2 & -1 & 0 & 1 & 2 & 3 \end{bmatrix}$$

$$y \quad \begin{bmatrix} 29 & 20 & 13 & 8 & 5 & 4 & 5 \end{bmatrix}$$

Czyli po rozwiązaniu zadania możemy powiedzieć ile wynosi wartość funkcji dla $x = -1$, albo dla $x = 2$, ale nie możemy precyzyjnie odpowiedzieć na pytanie ile wynosi wartość funkcji dla $x = 1,5$ ponieważ nie mamy takiego punktu w wektorze!

Natomiast wykres to tylko wizualizacja tego co mamy zapisane w tych dwóch wektorach. Wykresy nie dostarczają nam żadnej nowej informacji. Powyższe punkty zostają naniesione na wykres i połączone odcinkiem. Ale tak naprawdę nie wiemy co *dzieje się* pomiędzy punktami, możemy tylko z pewnym przybliżeniem oszacować wartość w punkcie, którego nie obliczyliśmy.

Więc co trzeba zrobić żeby znaleźć więcej rozwiązań w zakresie od -3:3?
Oczywiście *zagęścić* wektor **x**, zmniejszając krok!
Na przykład:

```
x=-3:0.1:3
```

Teraz oba wektory będą wyglądały tak (trochę je tutaj przyciąłem bo są za długie):

$$x \quad \begin{bmatrix} -3 & -2.9 & -2.8 & \dots & -0.2 & -0.1 & 0 & 0.1 & 0.2 & \dots & 2.8 & 2.9 & 3 \end{bmatrix}$$

$$y \quad \begin{bmatrix} 29 & 28.01 & 27.04 & \dots & 8.84 & 8.41 & 8 & 7.61 & 7.24 & \dots & 4.64 & 4.81 & 5 \end{bmatrix}$$

Teraz możemy odczytywać argumenty **x** z dokładnością do 0.1 oraz podawać dla nich obliczone wartości **y**.
Odczytaj z wektorów ile wynosi wartość funkcji dla $x = 1.5$.

Odpowiedź na podstawowe pytanie w zadaniu narzuca się sama. Teraz punkty na

wykresie będą tak blisko siebie, a odcinki je łączące tak krótkie, że spowoduje to złudzenie wykresu ciągłej funkcji. Jeżeli wektory są bardzo *gęste* to raczej nie zaznaczamy punktów markerami, ponieważ zaczną na siebie nachodzić (tworząc grubą linię) i lepiej narysować samą krzywą.

Informacja dodatkowa: Ktoś mógłby zapytać, to po co zastanawiać się nad krokiem? Lepiej od razu dać 0.000001!

Oczywiście gęstsze wektory to większa ilość danych, dłuższy czas obliczeń, większa ilość zajętej pamięci. W metodach numerycznych ważne jest znalezienie kompromisu pomiędzy dokładnością rozwiązania, a czasem obliczeniowym i gospodarowaniem zasobami sprzętowymi.

Zadanie 6: Dodaj do pętli licznik, który poda ile razy odbyło się losowanie liczby.

Czyli do pętli musimy wstawić jakąś zmienną, którą w każdym kroku będziemy powiększali o 1. W języku C mamy do tego polecenie `i++`, jednak w matlabie nie ma takiej funkcji. Stąd trzeba zrobić to w następujący sposób:

```
k=0
a=0
while a<=999
  a=1000*rand(1)
  k=k+1
end
```

Czyli stare **k** zostanie nadpisane nowym **k** powiększonym o 1.
Oczywiście **k** należy wcześniej zadeklarować i przypisać mu wartość początkową.

Zadanie 7: Napisać program, który będzie realizował następujące zadanie. Wygeneruje wektor **x** zawierający **20** losowych elementów, każdy przyjmujący wartość od 0 do 1. Należy sprawdzić każdy element wektora (np. za pomocą pętli FOR) i w zależności od tego w jakim przedziale znajduje się jego wartość zastąpić go elementem o innej wartości:
1. Jeżeli $x_k \leqslant 0.3$ to $x_k = 0$
2. Jeżeli $x_k > 0.8$ to $x_k = 1$
3. Jeżeli $0.3 < x_k \leqslant 0.8$ (czyli jedyna pozostała możliwość) to $x_k = x_k$ (pozostaje bez zmian).
Przykładowo jeśli element $x_k = 0.2379$ to znaczy że spełnia warunek 1. i należy go zastąpić liczbą 0.
Na końcu porównać oba wektory.

Najpierw zapiszmy program, potem go omówimy:

```
clear all
```

```
clc
x=rand(1,20)
for k=1:length(x)
    if x(k)<=0.3
       x(k)=0;
    elseif >0.8
       x(k)=1;
    else                  %te dwie linijki
       x(k)=x(k);         %można pominąć
    end
end
disp(x)
```

1. Utworzenie wektora **x** z losowymi elementami, zostało też opisane, przy odpowiedziach do zadań 2 i 3.
Pamietaj, że w nawiasie muszą znaleźć się dwie liczby, tutaj (1,20) określające rozmiar tablicy (1 wiersz i 20 kolumn). Jeżeli wpiszesz tylko (20) to otrzymasz kwadratową macierz 20x20.
2. Pętla **for** pozwala adresować kolejne elementy w wektorze. Zauważ, że liczbę elementów w wektorze **x** możemy sprawdzić automatycznie za pomocą polecenia **length**. Program sobie podstawi w to miejsce wartość 20.
3. Instrukcję warunkowa otwieramy poleceniem **if** i sprawdzamy pierwszy warunek, jeżeli jest spełniony to wykonujemy instrukcję, a jeżeli nie to przechodzimy dalej. Kolejne warunki sprawdzamy poleceniem **elseif**, może być ich więcej niż jeden. Uwaga **elseif** piszemy łącznie, ponieważ rozdzielenie na **else if** spowoduje podzielnie tej instrukcji na dwie. Polecenie **else** oznacza w każdym pozostałym przypadku. W tym zadaniu możemy pominąć ten fragment, ponieważ tak naprawdę nie dokonujemy żadnej zmiany.
4. Warto zauważyć, że na końcu pojawia się 2 razy **end**. Jeden *end* zamyka instrukcję **if**, natomiast drugi pętlę FOR.
5. Proszę zwrócić uwagę na rozmieszczenie średników. Przypominam, że w Matlabie średnik wyświetla działanie danej linijki lub nie (inne zastosowanie niż w języku C). Czyli w Command Window wyświetli się na ekranie tylko losowo utworzony wektor **x** i wektor **x** już po wszystkich zmianach, co umożliwi ich porównanie.
Jeżeli w instrukcji **if** pominiemy średniki to każda zmiana zostanie wyświetlona na ekarnie i trudno będzie porównać wektory przed i po zmianie.

10 Octave jako substytut Matlaba

Octave jest darmowym oprogramowaniem Open Source. Ma te same podstawowe funkcje oraz składnię co Matlab. Dlatego też, studenci w przypadku ograniczonej możliwości używania Matlaba, mogą z niego korzystać w domu.

Instalacja
Istnieje kilka metod, korzystania z Octave.
1. Można korzystać z Octave w wersji **online**. Po zalogowaniu się na stronie można przechowywać własne pliki skryptowe. Metoda ta niesie pewne ograniczenia np. limit dostępnej pamięci, co przy dużej liczbie pętli i bardziej złożonych programach może być problemem.
`http://octave-online.net/`

2. Instalacja pod systemem **Windows**.
Tutaj można znaleźć pliki instalacyjne:
`http://www.gnu.org/software/octave/download.html`

3. Instalacja pod systemem **Linux**. Wersja samego programu Octave pod Linuxem jest nieco łatwiejsza w obsłudze i instalacji niż pod Windowsem. Jeżeli ktoś nie ma na komputerze systemu Linux może zainstalować go na wirtualnej maszynie.
a) Zainstalować program umożliwiający uruchomienie wirtualnej maszyny np VirtualBox `https://www.virtualbox.org/`.
b) Na wirtualnej maszynie zainstalować system Linux, np. Ubuntu
`http://www.ubuntu.com/`.
c) Zainstalować program Octave wraz z GUI.

```
sudo apt-add-repository ppa:octave/stable
sudo apt-get update
sudo apt-get install octave
```

Wszelkie zastrzeżenia, błędy merytoryczne oraz propozycje przedstawienia wybranych informacji w sposób bardziej klarowny, uprzejmie proszę o kierowanie na adres e-mail kszopa [at] agh.edu.pl.