

Skrypty i funkcje.
(wspomaganie obliczeń inżynierskich)

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z tworzeniem i wywoływaniem skryptów i funkcji języka GNU octave.

Wprowadzenie

Skrypty

Skrypty są najprostszym rodzajem m-plików. Nie pobierają one argumentów wejściowych i nie zwracają argumentów wyjściowych. Skrypty są użyteczne w przypadku powtarzania cyklu poleceń, które w przeciwnym razie byłyby wykonywane z linii poleceń. Skrypty współdzielą przestrzeń roboczą ze środowiskiem interaktywnej linii poleceń oraz z innymi skryptami. Działają na zmiennych istniejących w przestrzeni roboczej lub tworzą nowe zmienne w tej przestrzeni. Każda zmienna utworzona przez skrypt pozostaje w pamięci po zakończeniu wykonania skryptu, więc jest możliwe wykorzystanie jej w dalszych obliczeniach. Należy uważać aby skrypt nie nadpisał zmiennych tworzonych z poziomu interfejsu linii poleceń.

Przykład. Prosty skrypt generujący wykres w kształcie płatka kwiatu.

```
% M-plik skryptowy do utworzenia          % Linie komentarza
% wykresu typu "płatki-kwiatu"
theta = -pi:0.01:pi;                       % Obliczenia
rho(1,:) = 2 * sin(5 * theta) .^ 2;
rho(2,:) = cos(10 * theta) .^ 3;
rho(3,:) = sin(theta) .^ 2;
rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
for k = 1:4
    polar(theta, rho(k,:))                 % Grafika wynikowa (wykres)
    pause
end
```

Kod należy zapisać w pliku *platki.m*. uruchomienie skryptu następuje przez wywołanie polecenia *platki*.
UWAGA: plik *platki.m* musi znajdować się na liście ścieżek programu GNU octave.

Funkcje

Funkcje standardowe

Funkcje są procedurami programowymi zazwyczaj zaimplementowanymi w postaci m-pliku, które mogą przyjmować argumenty wejściowe i zwracać argumenty wyjściowe (wartości). Działają one w swojej własnej przestrzeni roboczej. Przestrzeń ta jest oddzielona od przestrzeni dostępnej z wiersza poleceń.

Każda funkcja zajmuje pewien obszar pamięci, w którym działa odseparowany od przestrzeni roboczej środowiska GNU octave. Ten obszar pamięci zwany jest kontekstem funkcyjnym. Podczas wykonywania funkcja ma dostęp jedynie do tych zmiennych, które istnieją w jej kontekście. Można jednak zdefiniować zmienne jako globalne. W takim przypadku zmienne mogą występować w kilku różnych kontekstach przestrzeni roboczych. Poniżej podano przykładową funkcję wraz z typowymi elementami kodu.

Przykład.

```
function y = average(x)
% AVERAGE Średnia elementów wektora.
% AVERAGE(X), gdzie X jest wektorem, zwraca średnią wktora X.
% Niewektorowe argumenty wejściowe skutkują wystąpieniem błędu.
[m,n] = size(x);
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
    error('Wejście musi być wektorem')
end
y = sum(x)/length(x);      % Rzeczywiste obliczenia.
```

W języku GNU octave istnieje bardzo wiele specjalizowanych funkcji. Podstawowe funkcje matematyczne w języku podzielono na następujące kategorie:

- funkcje trygonometryczne, hiperboliczne i odwrotne do nich,
- funkcje logarytmiczne, wykładnicze, potęgowe i wielomiany,
- funkcje związane z różnymi reprezentacjami liczb zespolonych,
- funkcje zmiany układu współrzędnych.

Dla przykładu w Tab. 1 zestawiono funkcje trygonometryczne, hiperboliczne i odwrotne do nich.

Funkcja	Opis
funkcje trygonometryczne	
$\sin(z)$	sinus: $\sin(z) = \frac{e^{iz} - e^{-iz}}{2i}$, $z \in C$
$\cos(z)$	cosinus: $\cos(z) = \frac{e^{iz} + e^{-iz}}{2i}$, $z \in C$
$\tan(z)$	tangens: $\tan(z) = \frac{\sin(z)}{\cos(z)}$, $z \in C$
funkcje cyklometryczne	
$\text{asin}(z)$	arcus sinus: $\arcsin(z) = -i \log(iz + \sqrt{1 - z^2})$, $z \in C$
$\text{acos}(z)$	arcus cosinus: $\arccos(z) = -i \log(z + i\sqrt{1 - z^2})$, $z \in C$
$\text{atan}(z)$	arcus tangens: $\arctan(z) = -0,5i \log\left(\frac{1+iz}{1-iz}\right)$, $z \in C$
funkcje hiperboliczne	
$\sinh(z)$	sinus hiperboliczny: $\sinh(z) = \frac{e^z - e^{-z}}{2}$, $z \in C$
$\cosh(z)$	cosinus hiperboliczny: $\cosh(z) = \frac{e^z + e^{-z}}{2}$, $z \in C$
$\tanh(z)$	tangens hiperboliczny: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, $z \in C$
funkcje odwrotne do hiperbolicznych	
$\text{asinh}(z)$	arcus sinus hiperboliczny: $\text{arsinh}(z) = \log(z + \sqrt{z^2 + 1})$, $z \in C$
$\text{acosh}(z)$	arcus cosinus hiperboliczny: $\text{arccosh}(z) = \log(z + \sqrt{z^2 - 1})$, $z \in C$
$\text{atanh}(z)$	arcus tangens hiperboliczny: $\text{artanh}(z) = \frac{1}{2} \log\left(\frac{1+z}{1-z}\right)$, $z \in C$

Tab. 1. Funkcje trygonometryczne i hiperboliczne.

Argumenty funkcji

Podczas wywoływania funkcji, kod wywołujący (lub interfejs linii poleceń) wyposaża funkcję we wszystkie dane, których potrzebuje ona do działania poprzez listę argumentów. Wszystkie dane zwracane przez funkcję są przekazywane do środowiska wywołującego poprzez listę zwracanych wartości. GNU octave zawsze przekazuje dane poprzez wartość. Jeżeli funkcja ma modyfikować przekazywane zmienne, należy przekazać je w postaci argumentów wyjściowych.

Funkcje `nargin` i `nargout` zwracają odpowiednio liczbę argumentów wejściowych i wyjściowych funkcji. Następnie możliwe jest wykorzystanie instrukcji warunkowych i zróżnicowanie zadań w zależności od liczby argumentów.

Funkcje `varargin` i `varargout` pozwalają na obsługę funkcji o zmiennej liczbie argumentów wejściowych lub wyjściowych. GNU octave pakuje wszystkie wyspecyfikowane w wywołaniu parametry do postaci macierzy komórkowej. Każda z komórek macierzy może przechowywać dane o różnych rozmiarach i typach.

Ponieważ `varargin` zawiera wszystkie argumenty wejściowe funkcji w postaci macierzy komórkowej, niezbędne jest użycie indeksowania macierzy komórkowej aby wyodrębnić dane. Indeksowanie komórkowe składa się z dwóch komponentów:

- indeksu w nawiasach klamrowych, określającego komórkę macierzy (argument wejściowy);
- indeksu w nawiasach zwykłych, określającego dane wewnątrz komórki.

Jeżeli funkcja zezwala na stosowanie zmiennej liczby argumentów wyjściowych, kod funkcji musi „upakować” argumenty wyjściowe do postaci macierzy komórkowej. Aby określić wewnątrz funkcji liczbę argumentów wyjściowych podanych w czasie wywoływania należy użyć `nargout`.

Parametry `varargin` oraz `varargout` muszą wystąpić na ostatnim miejscu listy argumentów. Oznacza to, że w wywołaniu funkcji w pierwszej kolejności powinny wystąpić argumenty wymagane.

Typy funkcji

Podstawowe typy funkcji w języku GNU octave to:

- funkcje anonimowe,
- funkcje główne,
- funkcje zagnieżdżone,
- podfunkcje,
- funkcje prywatne,
- funkcje przeciążone.

Funkcje anonimowe (ang. *anonymous function*) umożliwiają szybkie tworzenie kodu funkcji bez m-pliku. Funkcje anonimowe mają następującą składnię:

```
fhandle = @(arglist) expr
```

Czynnik `expr` definiuje kod funkcji (ciało). Najczęściej składa się z dowolnego, pojedynczego wyrażenia. Czynniki `arglist` stanowi listę argumentów wejściowych rozdzielonych przecinkami. Znak `@` tworzy wskaźnik do funkcji.

Przykład. Prosta funkcja anonimowa.

```
sqr = @(x) x.^2; %definicja
a = sqr(5) %wywołanie
a = 25
```

Przykład. Funkcja anonimowa o dwóch argumentach wejściowych. W przykładzie założono, że zmienne `A` i `B` zostały uprzednio zdefiniowane.

```
sumAxBy = @(x, y) (A*x + B*y); %definicja
sumAxBy(5, 7) %wywołanie
```

Jeżeli funkcja nie pobiera żadnych argumentów wejściowych w miejscu argumentów należy pozostawić nawiasy zwykłe.

Możliwe jest tworzenie macierzy funkcji anonimowych. Wiele funkcji anonimowych można zapisać w macierzy komórkowej.

Przykład. Macierz funkcji anonimowych.

```
%definicja
A = {@(x)x.^2, @(y)y+10, @(x,y)x.^2+y+10}
A =
    @(x)x.^2    @(y)y+10    @(x,y)x.^2+y+10
%wywołanie
A{1}(4) + A{2}(7)
ans =
    33
%wywołanie dla dwóch argumentów
A{3}(4, 7)
ans =
    33
```

Pierwsza funkcja zdefiniowana w m-pliku nazywana jest funkcją główną (ang. *primary*). Za funkcją główną może być zdefiniowana dowolna liczba podfunkcji (ang. *subfunctions*) służących jako procedury wywoływane w ciele funkcji głównej. W większości sytuacji, jedyną funkcją, którą można wywołać z poziomu linii lub poprzez inną funkcję jest funkcja główna.

Możliwe jest również definiowanie funkcji w ciele innej funkcji. Tak definiuje się funkcje zagnieżdżone. Funkcje zagnieżdżone mają następującą składnię:

```
function x = A(p1, p2)
...
    function y = B(p3)
        ...
    end
...
end
```

Funkcje mogą być zagnieżdżane wielokrotnie.

Przykład. Funkcje wielokrotnie zagnieżdżone.

W przykładzie stworzono funkcję C zagnieżdżoną wewnątrz funkcji B, która jest zagnieżdżona w A.

```
function x = A(p1, p2)
...

    function y = B(p3)
    ...
        function z = C(p4)
        ...
        end
    ...
    end
...
end
```

Zasięg zmiennych funkcji wyższego poziomu obejmuje funkcje zagnieżdżone. Innymi słowy funkcje zagnieżdżone mogą korzystać ze zmiennych funkcji nadrzędnych względem nich.

M-pliki mogą zawierać kod więcej niż jednej funkcji. Dodatkowe funkcje wewnątrz m-pliku są nazywane podfunkcjami (ang. *subfunctions*). Każda podfunkcja zaczyna się swoją indywidualną linią definicji. Podfunkcje są definiowane bezpośrednio jedna za drugą. Tak długo jak długo funkcja główna występuje na początku m-pliku, podfunkcje mogą występować w m-pliku w różnej kolejności.

Przykład. Struktura m-pliku zawierającego podfunkcje.

```
function [avg, med] = newstats(u) % funkcja główna
% NEWSTATS wyznacza średnią i medianę z pomocą wewnętrznych funkcji.
n = length(u);
avg = mean(u, n);
med = median(u, n);

function a = mean(v, n) % podfunkcja
% Obliczenie średniej.
a = sum(v)/n;

function m = median(v, n) % podfunkcja
% Oblicza medianę.
w = sort(v);
if rem(n, 2) == 1
    m = w((n+1) / 2);
else
    m = (w(n/2) + w(n/2+1)) / 2;
end
```

Podczas wywoływania funkcji z m-pliku program najpierw sprawdza, czy jest to podfunkcja.

Funkcje prywatne są zapisywane w specjalnym katalogu o nazwie `private`. Funkcje prywatne mogą być wywoływane tylko przez funkcje i skrypty spełniające następujące warunki:

- funkcja, która wywołuje funkcję prywatną musi być zdefiniowana w m-pliku znajdującym się w katalogu bezpośrednio nadrzędnym do `private`.
- Skrypt, który wywołuje funkcję musi sam być wywoływany przez funkcję, która posiada dostęp do funkcji prywatnej zgodnie z powyższą regułą.

Ponieważ funkcje prywatne są niewidoczne poza swoim katalogiem prywatnym (i nadrzędnym) mogą przyjmować nazwy jak funkcje znajdujące się w innych katalogach.

Wywoływanie funkcji

Kiedy następuje pierwsze wywołanie funkcji z linii poleceń bądź wewnątrz innej funkcji, kod zostaje przetłumaczony (ang. *parking*) na pseudokod i umieszczony w pamięci, po to aby za każdym razem gdy funkcja zostanie wywołana nie tłumaczyć go ponownie. Pseudokod pozostaje w pamięci dopóki nie zostanie usunięty poleceniem `clear` lub do końca pracy z programem. Aby usunąć kod z przestrzeni roboczej należy wprowadzić polecenia: `clear nazwa_funkcji` – usuwa z pamięci funkcję o nazwie `nazwa_funkcji`; `clear functions` – usuwa z pamięci wszystkie skompilowane funkcje; `clear all` – usuwa z pamięci wszystkie zmienne i funkcje.

Wywołanie funkcji może nastąpić zarówno z linii poleceń jak i poprzez inną funkcję. W trakcie wywołania należy upewnić się, czy zostały określone niezbędne argumenty wejściowe wywołania w nawiasach zwykłych i argumenty wyjściowe w nawiasach kwadratowych. Wywołania funkcji są wrażliwe na zmianę wielkości liter. Wywołanie funkcji może posiadać jedną z form:

- Wywołanie jako polecenie
- Wywołanie jako funkcja

Wywołanie jako polecenie ma następującą składnię:

```
nazwafunkcji in1 in2 ... inN
```

Wywołanie w formie polecenia jest dużo prostsze, lecz nie jest możliwe uzyskanie wartości. W wywołaniach w formie poleceń, program traktuje argumenty wejściowe jako łańcuchy.

Przykład. Wywołanie funkcji w formie polecenia.

```
save mydata.mat x y z  
clear length width depth
```

Wywołanie jako funkcja wygląda bardzo podobnie jak w większości języków programowania wysokiego poziomu. Jedyną różnicą jest możliwość zwracania przez funkcję GNU octave wielu wartości. Wywołanie funkcji z jednym argumentem wyjściowym ma następującą postać:

```
out = nazwafunkcji(in1, in2, ..., inN)
```

Jeżeli funkcja ma zwracać wiele wartości, argumenty wyjściowe rozdziela się przecinkami i umieszcza w nawiasach kwadratowych.

```
[out1, out2, ..., outN] = nazwafunkcji(in1, in2, ..., inN)
```

Przykład. Wywołania funkcji przy jednym i wielu argumentach wyjściowych.

```
copyfile(srcfile, '..\mytests', 'writable')  
[x1, x2, x3, x4] = deal(A{:})
```


Zadania

1. Uruchomić program GNU octave.
2. Uruchomić program Word (lub inny edytor tekstu).
3. Tworzenie skryptów obliczających wartość funkcji. Jeśli skrypty rozpoczną się od żądania podania z klawiatury wartości argumentu (funkcja `input`) to można sprawdzić poprawność kodu.
 - a) Utworzyć i uruchomić skrypt obliczający wartość funkcji dla: $x=-5$, $x=5$ i $x=0$:

$$t(x) = \begin{cases} -1 & \text{dla } x < 0 \\ 0 & \text{dla } x = 0 \\ 1 & \text{dla } x > 0 \end{cases}$$

W edytorze m-plików wprowadzić:

```
clc;                               %Czyszczenie ekranu
x=input('Podaj x: ') %Wczytanie wartości zmiennej x
if x<0                               %Obliczenia...
    t=-1
elseif x==0
    t=0
else
    t=1
end
```

Zapisać skrypt na dysku. Wprowadzić:

```
x=-5
```

Uruchomić skrypt (wpisując jego nazwę w linii poleceń). Powtórzyć powyższe operacje dla pozostałych wartości zmiennej x . zaobserwować wyniki. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

- b) Obliczyć wartość funkcji:

$$h(t) = \begin{cases} t - 10 & \text{dla } 0 < t < 100 \\ 0.45t + 900 & \text{dla } t > 0 \end{cases}$$

poza programem GNU octave dla następujących wartości zmiennej t :

- I. $t = 5, h = ?$
- II. $t = 110, h = ?$

Utworzyć i uruchomić skrypt dla powyższych wartości zmiennej t . Porównać wyniki.

UWAGA!!! W celu sprawdzenia, czy zmienna t należy do określonego przedziału można wykorzystać operator iloczynu logicznego & (np. dla $-1 < x < 1$: $(x > -1) \& (x < 1)$)

- c) Obliczyć wartość funkcji:

$$t(y) = \begin{cases} 200 & \text{dla } y < 1000 \\ 200 + 0.1(y - 10000) & \text{dla } 10000 \leq y < 20000 \\ 1200 + 0.15(y - 20000) & \text{dla } 20000 \leq y < 50000 \\ 5700 + 0.25(y - 50000) & \text{dla } y \leq 50000 \end{cases}$$

Dla następujących wartości zmiennej y :

- I. $y = 5000, t = ?$
- II. $y = 17000, t = ?$
- III. $y = 25000, t = ?$
- IV. $y = 75000, t = ?$

Po wywołaniu skryptu dla wszystkich wartości parametru y skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word. Wyczyścić przestrzeń roboczą poleceniem `clear`.

4. Tworzenie skryptów z wykorzystaniem struktur sterujących.
 - a) Utworzyć i uruchomić skrypt tworzący macierz M o rozmiarze 4×4 wypełnioną wartością podaną przez użytkownika. Przetestować działanie skryptu dla następujących liczb: 33, 5, 0, $\pi/2$.

```
k=input('Podaj wartość: ');
for i=1:4
    for j=1:4
        M(i,j)=k
    end
end
end
```

Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

- b) Utworzyć i uruchomić skrypt tworzący macierz M o rozmiarze zadawanym przez użytkownika, wypełnioną wartością podaną przez użytkownika. Przetestować działanie skryptu dla następujących liczb: 33, 5, 0 oraz $\pi/2$ i rozmiarów 2×2 oraz 7×2 . Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.
- c) Utworzyć skrypt tworzący macierz A o rozmiarze zadawanym przez użytkownika o elementach:

$$A_{ij} = \sqrt{1 + \frac{i}{j}}$$

- d) Uzupełnić w wykropkowanych miejscach i uruchomić skrypt rozmieniający zadaną przez użytkownika sumę pieniędzy na dostępne nominały (50, 20, 10, 5, 2, 1 zł, pominąć grosze).

```
....=input('Podaj sumę: ') %Wczytanie danych
liczba_nominal=[0 0 0 0 0 0]; %Zmienne pomocnicze
nominaly=[50 20 10 5 2 1];
nominal_index=1;
while x....0 %Warunek zakończenia pętli
    liczba_nominal(...)=fix(x./nominaly(nominal_index));
    pozostalo=mod(x, ....(nominal_index))
    x=pozostalo;
    nominal_index=nominal_index+....
end
liczba_nominal
```

Przetestować działanie skryptu dla następujących wartości: 100 i 120. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

- e) Napisać skrypt do obliczania reszty z banknotu 100 zł w dostępnych nominałach (50, 20, 10, 5, 2 i 1, pominąć grosze) po zakupie na kwotę mniejszą od 100 zł. Resztę rozpisywać na największe

dostępne nominały. Program ma wczytywać kwotę zakupu $x < 100$ i podawać liczbę każdego z nominałów reszty. Wykorzystać kod skryptu utworzonego w punkcie poprzednim. Przetestować program dla $x = \{1, 12, 51\}$. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

f) Dany jest wektor $x = [1 \ 8 \ 3 \ 9 \ 0 \ 1]$. Napisać i uruchomić skrypt do obliczania:

- I. Sumy wartości elementów x (wykorzystać pętlę `for`),
- II. Iloczyn wartości elementów x (wykorzystać pętlę `while`).

Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

Utworzyć skrypt obliczający jaka jest największa wartość liczby n , która może być użyta w sumie: $1^2 + 2^2 + \dots + n^2$, aby suma ta nie przekroczyła 100. Wykorzystać pętlę `while`. Przetestować skrypt dla sumy nieprzekraczającej 1230. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

g) Uzupełnić w wy kropkowanych miejscach i uruchomić skrypt rozwiązujący równanie: $x = \cos x$. Przetestować skrypt dla dokładności $d = 0.001$. Algorytm rozwiązania można podzielić na następujące etapy:

- I. założenie wartości początkowej,
- II. wykonanie iteracji zgodnie ze schematem: $x_{n+1} = \cos(x_n)$,
- III. iteracje kończą się gdy następuje warunek: $|x_{n+1} - x_n| \geq \varepsilon$, gdzie ε - zadana dokładność.

```
x = zeros(1,20); x(1) = pi/4; n = ....; d = 1;
while d ....
    n = n+....;
    x(n) = cos(x(n-1));
    d = abs( .... - x(n-1) );
end
```

Przetestować działanie dla $d = 10^{-6}$. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

5. Tworzenie prostych funkcji.

a) Utworzyć funkcję obliczającą wartość wyrażenia: $y = \ln(2 + t + t^2)$, pobierającą od użytkownika wartość t w postaci argumentu wejściowego i zwracającą wynik do zmiennej y . wywołać funkcję z linii poleceń. W celu realizacji zadania utworzyć następujący m=plik:

```
function y=funkcja_1(t) % Nagłówek t-wejście, y-wyjście
% Funkcja oblicza wartość wyrażenia y=ln(2+t+t^2) (Linia H1)
y=log(2+t+t^2);
```

Zapisać plik pod nazwą funkcja_1.m. W celu wywołania funkcji dla $t = 1$ wprowadzić z linii poleceń:

```
>>wynik_f1=funkcja_1(1)
```

Przetestować działanie dla $t = 10$, $t = \pi/2$, $t = 0.01$. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

- b) Utworzyć funkcję obliczającą wartość wyrażenia: $y = \frac{10t_1}{3t_1+t_2}$, pobierającą od użytkownika wartości t_1, t_2 w postaci argumentów wejściowych i zwracającą wyniki do zmiennej y . Wywołać funkcję z linii poleceń. W celu realizacji zadania utworzyć następujący m-plik:

```
function y=funkcja_2(t1,t2) %Nagłówek t1,t2-wejścia, y-wyjście
y=(10*t1)/(3*t1+t2);
```

Zapisać plik pod nazwą funkcja_2.m. w celu wywołania funkcji dla $t_1=1$ i $t_2=1$ wprowadzić z linii poleceń:

```
>>wynik_f2=funkcja_2(1,1)
```

Przetestować działanie funkcji dla $t_1=2$ i $t_2=5$, $t_1=\pi/2$ i $t_2=\pi$, $t_1=0.1$ i $t_2=0.01$. Skopiować zawartość okna edytora oraz wyniki działania skryptu do programu Word.

- c) Utworzyć funkcje obliczające wartości następujących wyrażen:

I. $y = a + \frac{ab(a+b)^2}{c(a+b+c)}$ a, b, c – argumenty wejściowe

II. $M = \frac{2}{3} \log \frac{E}{E_0}$ E, E_0 – argumenty wejściowe

III. $A_m = P(1 + \frac{r}{m})^{mt}$ A_m, P, r, m, t – argumenty wejściowe

Każdą z utworzonych funkcji wywołać z linii poleceń dla dwóch dowolnych zestawów parametrów wejściowych. Skopiować listingi oraz wyniki działania każdej funkcji do programu Word.

- d) Utworzyć funkcję obliczającą pierwiastki trójmianu kwadratowego $y = ax^2 + bx + c$ dla zadanych argumentów wejściowych a, b i c . do wyboru rozwiązania w zależności od znaku wyróżnika trójmianu wykorzystać instrukcję warunkową `if`. Funkcja powinna zwracać dwie wartości (pierwiastki). Przetestować funkcję dla następujących zestawów parametrów wejściowych:

I. $a = 1, b = 5, c = 5$

II. $a = 2, b = 2, c = 3$

III. $a = 1, b = 2, c = 1$

Skopiować listing funkcji oraz wyniki jej działania do programu Word.

- e) Utworzyć funkcję obliczającą zadaną liczbę wyrazów ciągu Fibonacciego. Funkcja powinna umożliwiać wybór metody obliczeń. Metoda pierwsza wykorzystuje wzór rekurencyjny: $x_{k+1} = x_k + x_{k-1}$ dla $x_0=x_1=1$. Druga metoda korzysta ze wzoru:

$$x_k = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^k - \left(\frac{1 - \sqrt{5}}{2} \right)^k \right], \quad k = 0, 1, 2, \dots$$

Przykładowy nagłówek funkcji:

```
function [x]=fibb(n,metoda)
```

Przetestować działanie funkcji (obie metody) dla:

- I. $n = 10$
- II. $n = 100$
- III. $n = 10^8$

Skopiować listing funkcji oraz wyniki jej działania do programu Word.

- f) Utworzyć funkcję obliczającą ile liczb pierwszych znajduje się w podanym zakresie. Do sprawdzenia, czy dana liczba x jest liczbą pierwszą można wykorzystać funkcję $\text{rem}(x, 2)$, przetestować działanie funkcji dla:

- I. $x = 10$
- II. $x = 100$
- III. $x = 100000$

Skopiować listing funkcji oraz wyniki jej działania do programu Word.

- g) Rezystancja R [Ω] przewodu o długości l [m] i polu przekroju poprzecznego s [m²] wykonanego z materiału o rezystywności ρ [Ωm] dana jest wzorem: $R = \frac{\rho l}{s}$. Napisać funkcję wyznaczającą wartość rezystancji na podstawie trzech argumentów wejściowych występujących w powyższym wzorze. Przetestować działanie funkcji obliczając rezystancję przewodu o długości $l=5m$ o przekroju $s=2 \text{ mm}^2$, wykonanego z:

- I. miedzi: $\rho = 1,7 \cdot 10^{-8} \Omega m$
- II. aluminium: $\rho = 2,82 \cdot 10^{-8} \Omega m$

UWAGA: Pole przekroju poprzecznego należy przeliczyć na m². Skopiować listing funkcji oraz wyniki jej działania do programu Word.