Informatyka

Wykład 1

Witold Dyrka
witold.dyrka@pwr.wroc.pl

20/2/2012

Program wykładów

0. Informatyka. Wprowadzenie do Matlaba (13.02.12)
1. **Matlab dla programistów C/C++** (20.02.12)
2. Optymalizacja obliczeń. Grafika w Matlabie (05.03.12)
3. Programowanie zorientowane obiektowo (19.03.12)
4. Programowanie zorientowane obiektowe w praktyce (02.04.12)
5. Graficzny interfejs użytkownika (16.04.12)
6. Obliczenia numeryczne (30.04.12)
7. Kolokwium (14.05.12)

Dzisiejszy wykład w oparciu o...

- B. Mrozek, Z. Mrozek. MATLAB i Simulink. Poradnik użytkownika. Wydanie III. Helion 2010. Rozdział 3 i 5.
- MATLAB® Getting Started Guide (R2011b). http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
- MATLAB Documentation Center (beta). Programming and Data Types http://www.mathworks.com/help/matlab/
- R. Sedgewick, K. Wayne. Introduction to Programming in Java. Appendix E: MATLAB. http://introcs.cs.princeton.edu/java/11matlab/

Program na dziś

- **Uwarunkowania i cele – przypomnienie**
- Skrypty i funkcje
- Instrukcje sterujące
- *Typy danych (patrz wykład 1.5)*

Uwarunkowania

- Oczekuję, że potrafią Państwo:
  - opisać algorytm w formie pseudokodu i/lub schematu blokowego
  - korzystać z instrukcji warunkowych i pętli
  - dzielić program na funkcje, przekazywać parametry funkcji
  - formatować kod (np. wcięcia) oraz pisać komentarze
  - tworzyć własne typy danych: struktury C i/lub klasy C++ (przynajmniej niektórzy z Państwa)

Cele

- Rozwinięcie umiejętności poprawnego programowania:
  - programowanie proceduralne
  - elementy programowania obiektowego
  - tworzenie graficznego interfejsu użytkownika
- Nabycie umiejętności samodzielnego uczenia się
  - języka programowania
- Poznanie MATLABa jako narzędzia
  - rozwiązywania problemów numerycznych

Program na dziś (2)

- Uwarunkowania i cele – przypomnienie
- **Skrypty i funkcje**
  - M-skrypty
  - M-funkcje
    - funkcje główna
    - subfunkcje
    - przekazywanie parametrów
    - funkcje zagnieżdżone
      - widoczność i zakresy
- Instrukcje sterujące

„Pierwszy skrypt” w Matlabie

```
% Skrypt pitagoras1.m          na podst. Mrozek&Mrozek'10
% Warunki początkowe:
%   a, b - długości przyprostokątnych tr. prostokątnego
% Program oblicza długość przeciwprostokątnej.

c2 = a.^2 + b.^2;    % inicjacja zmiennej c2
c = sqrt(c2)         % inicjacja zmiennej c pierwiastkiem c2
                     % oraz wypisanie na ekranie
```

```
>> a=3; b=4;         % definicja zmiennych używanych w skrypcie
>> pitagoras1        % nazwa pliku bez rozszerzenia
c =
     5
```

- Nie ma funkcji **main**
- Skrypt korzysta z przestrzeni roboczej, z której jest wywoływany
  - `a` i `b` muszą być w przestrzeni roboczej (*Workspace*)
  - `c` i `c2` są umieszczane w przestrzeni roboczej

M-skrypt

- MATLAB wykonuje polecenia znalezione w skrypcie
  - tak jakby zostały podane w oknie poleceń (*Command Window*)
  - program skryptowy nie stanowi samodzielnej aplikacji
    - jest wywoływany wewnątrz innego programu, **interpretera**
    - nie jest kompilowany

- Skrypt nie przyjmuje argumentów ~~`% pitagoras1(a,b)`~~
  - ale operuje na danych z aktualnej przestrzeni roboczej
- Skrypt niczego nie zwraca zwraca ~~`% c = pitagoras1()`~~
  - ale utworzone zmienne pozostają w przestrzeni roboczej

„Pierwsza funkcja” w Matlabie

```
function c=pitagoras(a,b)
% PITAGORAS na podst. Mrozek&Mrozek'10
% Wywołanie: c = pitagoras(a,b)
%   a, b - długości przyprostokątnych trójkąta prostokątnego
%   c    - długość przeciwprostokątnej.

c2 = a.^2 + b.^2;    % inicjacja zmiennej c2
c = sqrt(c2);        % inicjacja zmiennej c pierwiastkiem c2
                     % oraz wypisanie na ekranie

end                  % niekonieczne
```

```
>> c = pitagoras(a,b)      % nazwa pliku bez rozszerzenia
c =
     5
```

- Funkcja powinna mieć taką samą nazwę jak plik w którym się znajduje
- Funkcja ma własną przestrzeni roboczą (zakres lokalny)
  - `a` i `b` muszą zostać przekazane jako parametry, a `c` zostać zwrócone
  - `c` i `c2` żyją tylko w lokalnej przestrzeni roboczej funkcji

Definicja M-funkcji

```
function [ argumenty_wy ] = Nazwa_funkcji( argumenty_we )
%NAZWA_FUNKCJI Miejsce na krótki opis
%   Miejsce na dokładny opis

% kod funkcji ...

end
```

- Funkcja może przyjąć wiele argumentów wejściowych
  - i zwrócić wiele argumentów wyjściowych

- Opis funkcji pod nagłówkiem wykorzystuje polecenie `help`:

```
>> help Nazwa_funkcji
 Nazwa_funkcji Miejsce na krótki opis
    Miejsce na dokladny opis

>> help pitagoras
  pitagoras na podst. Mrozek&Mrozek'10
  Wywolanie: c = pitagoras(a,b)
    a, b - dlugosci przyprostokatnych trójkata prostokatnego
    c    - dlugosc przeciwprostokatnej
```

MATLAB – język interpretowany

- Programy w MATLABie nie wymagają funkcji **main**
  - dlaczego?
  - typowo nie tworzymy samodzielnych aplikacji
    - funkcje są wywoływane wewnątrz interpretera
    - mówimy, że MATLAB nie kompiluje kodu, ale go interpretuje

- Dzięki interpretowaniu łatwiej uzyskać:
  - niezależność od platformy sprzętowej
  - dynamiczny systemu typów
  - dynamiczny zakres

- Kosztem użycia interpretera jest gorsza wydajność

Funkcja główna

- Pierwsza funkcja w pliku jest funkcją główną
  - tylko funkcję główną można wywołać spoza tego pliku
  - nazwa pliku i funkcji głównej powinna być zgodna
    - interpreter rozpoznaje funkcje po nazwie pliku

    np. zapiszmy naszą funkcję `pitagoras` w pliku **tales.m:**

```
function c=pitagoras(a,b)
% PITAGORAS na podst. Mrozek&Mrozek'10
% ...
c2 = a.^2 + b.^2;
c = sqrt(c2);
```

wywołujemy w oknie poleceń:

```
>> c = tales(a,b)
c =
     5

>> help tales
  PITAGORAS na podst. Mrozek&Mrozek'10
  ...
```

Subfunkcje

- Kolejne funkcje w pliku są subfunkcjami
  - widzialne tylko dla funkcji głównej – służą jako funkcje pomocnicze
  - mają lokalne przestrzenie robocze – wymagają przekazywania argumentów

```
function [sr, med] = statystyka(u) % Funkcja glowna
% STATYSTYKA Znajduje srednia i mediana uzywajac funkcji wewnetrznych
n = length(u);                     % dlugosc wektora
sr = srednia(u, n);
med = mediana(u, n);

function a = srednia(v, n)         % Subfunkcja
% Liczy srednia.
a = sum(v)/n;                      % suma elementow wektora / dlugosc

function m = mediana(v, n)         % Subfunkcja
% Liczy mediane.
w = sort(v);                       % sortowanie wektora
if rem(n, 2) == 1                  % reszta z dzielenia
   m = w((n+1) / 2);
else
   m = (w(n/2) + w(n/2+1)) / 2;
end
```

http://www.mathworks.com/help/matlab/matlab_prog/subfunctions.html

Subfunkcje – przykład

```
function [sr, med] = statystyka(u) % Funkcja glowna
% STATYSTYKA Znajduje srednia i mediana uzywajac funkcji wewnetrznych
n = length(u);                     % dlugosc wektora
sr = srednia(u, n);
med = mediana(u, n);

function a = srednia(v, n)         % Subfunkcja
% Liczy srednia.
a = sum(v)/n;                      % suma elementow wektora / dlugosc
```

```
>> statystyka([1 4 8])             % jeśli nie podano listy parametrów wy
ans =                              % zwróci tylko pierwszy
    4.3333

>> s = statystyka([1 4 8])
s =
    4.3333

>> [s m] = statystyka([1 4 8])     % odebranie dwu zwróconych wartości
s =
    4.3333
m =
     4

>> help statystyka>srednia         % dostęp do pomocy subfunkcji
  Liczy srednia.
```

Przekazywanie parametrów do/z funkcji

- MATLAB zawsze przekazuje argumenty przez wartość
  - funkcja posiada własną kopię parametru
    - kopia dużych macierzy jest tworzona dopiero w momencie modyfikacji zmiennej
  - zwracanie parametrów też wymaga kopiowania

- Używanie zmiennych globalnych jest niezalecane

- Uniknąć kopiowania dużych macierzy można stosując funkcje zagnieżdżone

Funkcje zagnieżdżone

```
function y = taxDemoPL(income)
%TAXDEMOPL jest wersją taxdemo używanego przez nesteddemo.
% Oblicza podatek dochodowy w warunkach polskich.
% Użyj polecenia: type taxdemo, żeby zobaczyć oryginalne taxdemo

AdjustedIncome = income - 3091;   % Dochód – kwota wolna

% Wywołanie computeTax nie wymaga przekazania parametru AdjustedIncome
y = computeTax;

    % Funkcja zagnieżdżona
    function y = computeTax

        % Funkcja computeTax widzi zmienną AdjustedIncome,
        % która znajduje się w przestrzeni roboczej funkcji wywołującej

        y = 0.18 * AdjustedIncome;

    end % Funkcja zagnieżdżona musi być zakończona end-em

end % Funkcja zawierająca funkcję zagnieżdżoną musi być zakończona end-em
```

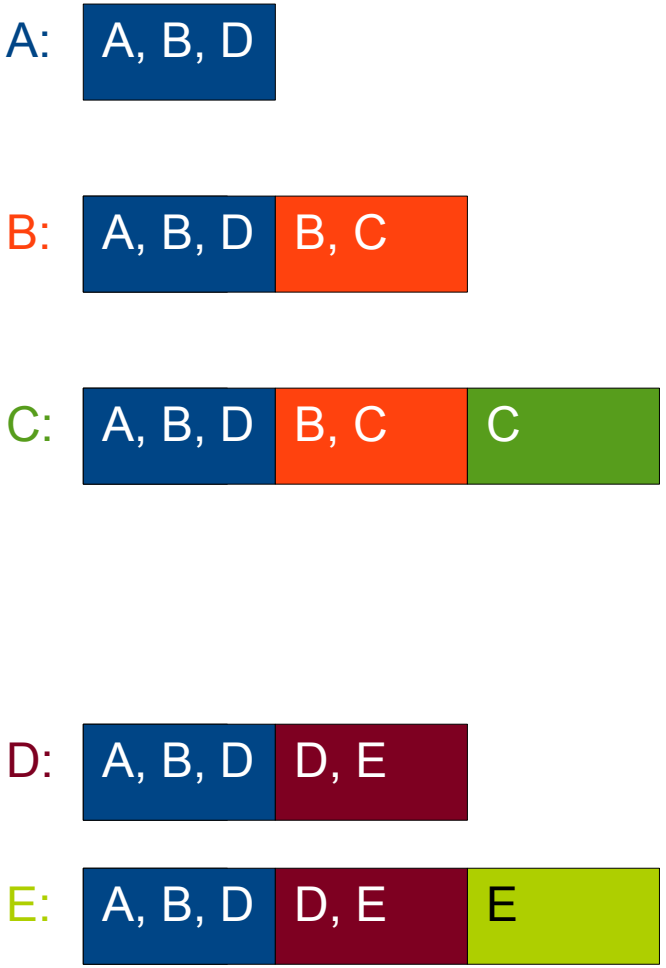
```
>> taxDemoPL(25000)
ans =
  3.9436e+003          % ups, kiepski format
>> format short g      % optymalny (stało/zmiennoprzecinkowy), 5 cyfr
>> taxDemoPL(25000)
ans =
       3943.6
```

Widoczność funkcji zagnieżdżonych

```
function A(x, y)          % Funkcja główna

B(x, y);                  % - widzi tylko funkcje
D(y);                     %   zagnieżdżone w niej bezpośrednio

   function B(x, y)       % Funkcja zagnieżdżona w A

   C(x);                  % - widzi funkcję C zagnieżdżoną w niej
   D(y);                  % - i D zagnieżdżoną na tym samym poziomie co ona sama

      function C(x)       % Funkcja zagnieżdżona w B

      D(x);               % - widzi funkcję D, bo była widoczna z B

      end

   end

   function D(x)          % Funkcja zagnieżdżona w A

    E(x);                 % - widzi funkcję E zagnieżdżoną w niej

      function E(x)       % Funkcja zagnieżdżona w D

      %...

      end

   end
end
```

Widoczność funkcji

```
function A(x, y)

B(x, y);
D(y);

    function B(x, y)

    C(x);
    D(y);

        function C(x)

        D(x);

        end

    end

    function D(x)

     E(x);

        function E(x)

        %...

        end

    end
end
```



Zakresy zmiennych w funkcjach zagnieżdżonych

- Zmienna używana lub definiowana w funkcji zagnieżdżonej znajduje się w przestrzeni roboczej zewnętrznej funkcji, która
  - zawiera tę funkcję zagnieżdżoną
  - korzysta z tej zmiennej

```
function varScope1
x = 5;
nestfun1
   function nestfun1
      nestfun2

      function nestfun2
         x = x + 1
      end
   end
end
```

- `x` w przestrzeni roboczej funkcji `varScope1`

http://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html

Zakresy zmiennych w funkcjach zagnieżdżonych

- Zmienna używana lub definiowana w funkcji zagnieżdżonej znajduje się w przestrzeni roboczej zewnętrznej funkcji, która
  - zawiera tę funkcję zagnieżdżoną
  - korzysta z tej zmiennej

```
function varScope1
x = 5;
nestfun1
   function nestfun1
      nestfun2

      function nestfun2
         x = x + 1
      end
   end
end
```

- `x` w przestrzeni roboczej funkcji `varScope1`

```
function varScope2
nestfun1
   function nestfun1
      nestfun2

      function nestfun2
         x = 5;
      end
   end
x = x + 1
end
```

- `x` w przestrzeni roboczej funkcji `varScope2`, która
  - zawiera `nestfun2`
  - korzysta z `x`

http://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html

Zakresy zmiennych w funkcjach zagnieżdżonych

- Zmienna używana lub definiowana w funkcji zagnieżdżonej znajduje się w przestrzeni roboczej zewnętrznej funkcji, która
  - zawiera tę funkcję zagnieżdżoną
  - korzysta z tej zmiennej

```
function varScope1
x = 5;
nestfun1
   function nestfun1
      nestfun2

      function nestfun2
         x = x + 1
      end
   end
end
```

- `x` w przestrzeni roboczej funkcji `varScope1`

```
function varScope2
nestfun1
   function nestfun1
      nestfun2

      function nestfun2
         x = 5;
      end
   end
x = x + 1
end
```

- `x` w przestrzeni roboczej funkcji `varScope2`, która
  - zawiera `nestfun2`
  - korzysta z `x`

```
function varScope3
nestfun1
nestfun2

   function nestfun1
      x = 5;
   end

   function nestfun2
      x = x + 1
   end
end
```

- dwa różne `x` w funkcjach `nestfun1` i `nestfun2`
  - stąd błąd w `nestfun2`

http://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html

Zakres zmiennych wyjściowych w funkcjach zagnieżdżonych

- Parametry wyjściowe zwracane przez funkcję zagnieżdżoną znajduje się w lokalnej przestrzeni roboczej tej funkcji

```
function varScope4
x = 5;
nestfun;

   function y = nestfun
      y = x + 1;
   end

y
end
```

- błąd!
  `y` niezdefiniowane
  w przestrzeni roboczej
  funkcji `varScope4`

```
function varScope5
x = 5;
z = nestfun;

   function y = nestfun
      y = x + 1;
   end

z
end
```

- dobrze

http://www.mathworks.com/help/matlab/matlab_prog/nested-functions.html

Program na dziś (3)

- Uwarunkowania i cele – przypomnienie
- Skrypty i funkcje
- **Instrukcje sterujące**
  - instrukcje warunkowe
    - operatory porównania
  - pętle
  - obsługa błędów
    - sprawdzanie warunków początkowych

Instrukcje języka MATLAB

- Instrukcje warunkowe
  - `if - elseif - else - end`
  - `switch - case - otherwise - end`
- Pętle
  - `while - end`
  - `for - end`
  - `continue, break`
- Obsługa błędów
  - `try-catch-end`

Instrukcje warunkowe:
if – elseif – else – end

```
twojaLiczba = input('Podaj liczbe: ');    % czyta zmienną z konsoli
                                          % w Matlabie literał łańcuchowy
                                          % w „uszach”, a nie w cudzysłowie

% Instrukcja warunkowa if:

if twojaLiczba < 0          % warunek nie musi być w nawiasie

    disp('Ujemna')          % wyświetla zmienną: tutaj łańcuch

elseif twojaLiczba > 0      % elseif to jedno słowo kluczowe

    disp('Dodatnia')

else

    disp('Zero')

end                         % instrukcję warunkową kończymy end
```

http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf, s.157

Instrukcje warunkowe:
switch – case – otherwise – end

```
[numerDnia, nazwaDnia] = weekday(date, 'long');  % date-funkcja zwracająca
                                                 % aktualna datę
                                                 % date ≡ date()
                        % weekday - funkcja zwracająca dzień tygodnia
                        % dla podanej daty i zadanego formatu (tu:'long')

% Instrukcja wyboru switch:   w przeciwieństwie do C/C++
%                             wykonuje tylko jeden przypadek
%                             nie trzeba break

switch nazwaDnia   % wartość wyrażenia sterującego wyborem (tu: nazwaDnia)
                   % musi być skalarem lub łańcuchem

    case 'Monday'                                % pojedyncza stała
        disp('Pierwszy dzien tygodnia pracy')

    case {'Tuesday','Wednesday','Thursday'}      % lub lista stałych
        disp('Środek tygodnia pracy')

    case 'Friday'
        disp('Ostatni dzien tygodnia pracy')

    otherwise                                 % to samo co default: w C/C++
        disp('Weekend!')

end
```

Operatory porównania

`A<B, A>B, A<=B, A>=B, A==B, A~=B`

- porównują odpowiadające sobie elementy macierzy:

```
>> A = rand(3), B=rand(3)
A =
    0.8147    0.9134    0.2785
    0.9058    0.6324    0.5469
    0.1270    0.0975    0.9575

B =
    0.9649    0.9572    0.1419
    0.1576    0.4854    0.4218
    0.9706    0.8003    0.9157

>> A>B
ans =
     0     0     1
     1     1     1
     0     0     1
```

`A<B, A>B, A<=B, A>=B` – tylko części rzeczywiste liczb zespolonych

`A==B, A~=B` – część rzeczywistą i urojoną

Porównywanie tablic w instrukcjach warunkowych

- Przykład: chcemy sprawdzić czy tablice są różne:

```
>> A = ones(2), B=ones(2); B(1,1)=0
A =
     1     1
     1     1

B =
     0     1
     1     1

>> if (A~=B) disp('Rozne'), else disp('Identyczne'), end
Identyczne                                                    % Co???
       % operator ~= zwraca tablicę wartości logicznych NxN (tu:2x2)

>> if (~isequal(A,B)) disp('Rozne'), else disp('Identyczne'), end
Rozne                                                         % Ok.
       % funkcja isequal zwraca skalarną wartość logiczną (1/0)
```

Pętle:
`while - end`

- składnia podobna jak w C/C++

```
function v = potega2(N)
%POTEGA2 zwraca największa potęgę 2-ki mniejsza od zadanego N

v = 1;

while v <= N/2

   v = 2 * v;

end
```

- nie ma pętli `do - while`

Pętle:
`for - end`

```
function p = silnia(N)
%SILNIA zwraca iloczyn 1 * 2 * ... * N

p = 1;

for k = 1:N          % w C/C++:  for (int k=1; k<=N; k++)

   p = p * k;

end
```

- nie należy iterować po `i` i `j`,
  - w programach działających na liczbach zespolonych `i = j = sqrt(-1)`

Pętle: `for - end` (2)

- ogólna postać pętli `for`:

```
for zmienna_iteracyjna = wartość_początkowa: krok : wartość_końcowa
    polecenia
end
```

Pętle: `for - end` (2)

- ogólna postać pętli `for`:

```
for zmienna_iteracyjna = wartość_początkowa: krok : wartość_końcowa
    polecenia
end
```

- np.

```
for x = pi/100 : pi/100 : 10*pi;
    y = sin(x)/x;
    plot(x, y);
    hold on;            % następny wykres nie zamaże poprzedniego
end
```

Pętle: `for - end` (2)

- ogólna postać pętli `for`:

```
for zmienna_iteracyjna = wartość_początkowa: krok : wartość_końcowa
    polecenia
end
```

- np.

```
for x = pi/100 : pi/100 : 10*pi;
    y = sin(x)/x;
    plot(x, y);
    hold on;            % następny wykres nie zamaże poprzedniego
end
```

- nieefektywny kod, lepiej zrobić to tak:

```
x = pi/100 : pi/100 : 10*pi;   % = linspace(pi/100, 10*pi, 1000)

y = sin(x)./x;                 % tablicowe dzielenie sinusa

plot(x, y, '.');               % wykreśla same punkty
```

- jest to przykład <u>wektoryzacji</u> pętli

Przykład

```
function liczba_linii = liniekodu(nazwa_pliku)
%LINIEKODU zlicza liczbę linii kodu w pliku źrodłowym
% Wywołanie: liczna_linii = liniekodu(nazwa_pliku)

fid = fopen(nazwa_pliku,'r');    % otwiera plik do odczytu w trybie bin.
count = 0;

while ~feof(fid)                 % dopóki nie natrafi na koniec pliku

    line = strtrim(fgetl(fid)); % fgetl czyta linie; strtrim usuwa wcięcia

    if isempty(line) || strncmp(line,'%',1) || ~ischar(line)
                                 % - jeśli linia pusta lub komentarz
                                 % lub nie jest tablicą znaków
        continue                 % - przejdź do początku pętli pomijając
                                 % pozostałe polecenia
    end

    count = count + 1;           % zwiększ licznik linii kodu
end

fprintf('%d linii\n',count);     % formatowane wyjście - podobnie jak w C
fclose(fid);                     % zamyka plik
```

```
>> liniekodu('liniekodu.m')
12 linii
```

http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf, s.161

Przykład (2)

```
function [nr_ln, txt_ln] = pierWystNazwy(nazwa_pliku, szukana_nazwa)
%PIERWYSTNAZWY szuka linii kodu, w której po raz pierwszy użyto nazwy
% Wywołanie: [nr_ln, txt_ln] = pierWystNazwy(nazwa_pliku, szukana_nazwa)

fid = fopen(nazwa_pliku,'r');   % otwiera plik do odczytu w trybie bin.
nr_ln = 0;

while ~feof(fid)

   txt_ln = fgetl(fid);
   nr_ln = nr_ln + 1;

   if strfind(txt_ln, szukana_nazwa)  % szukanie łańcucha w łańcuchu

       break                          % przerywa wykonanie pętli

   end
end

fclose(fid);
```

```
>> [n,l]=pierWystNazwy('liniekodu.m','strtrim')
n =
     8
l =
    line = strtrim(fgetl(fid)); % fgetl-czyta linie; strtrim-usuwa wciecia
```

Sprawdzenie warunków początkowych

- Instrukcja warunkowa + funkcja `error(id_błędu, komunikat_błędu)`

```
function C = mnozenieTablic(A, B)
%MNOZENIETABLIC mnozy dwie tablice, element po elemencie

if ~isequal(size(A),size(B))

   error('mnozenieTablic:wymiary',...                    % id_błędu
       'Wymiary macierzy A: %d x %d, B: %d x %d.',...    % komunikat_błędu:
       size(A,1), size(A,2), size(B,1), size(B,2));      % format jak printf
end

C=A.*B;
```

Sprawdzenie warunków początkowych

- Instrukcja warunkowa + funkcja `error(id_błędu, komunikat_błędu)`

```
function C = mnozenieTablic(A, B)
%MNOZENIETABLIC mnozy dwie tablice, element po elemencie

if ~isequal(size(A),size(B))

   error('mnozenieTablic:wymiary',...                    % id_błędu
       'Wymiary macierzy A: %d x %d, B: %d x %d.',...    % komunikat_błędu:
       size(A,1), size(A,2), size(B,1), size(B,2));      % format jak printf
end

C=A.*B;
```

- lub bardziej zwięźle – `assert(warunek_popr, komunikat_błędu)`

```
function C = mnozenieTablic(A, B)

assert(isequal(size(A),size(B)),...                  % warunek_poprawności
    'Wymiary macierzy A: %d x %d, B: %d x %d.',...   % komunikat_błędu
    size(A,1),size(A,2),size(B,1),size(B,2));

C=A.*B;
```

```
>> mnozenieTablic(A,B)
Error using mnozenieTablic (line 3)
Wymiary macierzy A: 2 x 2, B: 3 x 3.
```

Programowa obsługa błędów: `try-catch-end`

- Funkcje `error` oraz `assert` rzucają wyjątki
  - MATLAB sam wyświetla komunikaty o błędach
    - jeśli wyjątek nie został obsłużony

- Programista może zmienić komunikat
  - lub obsłużyć wyjątek:

```
function C = mnozenieWektorow(A, B)
%MNOZENIEWEKTOROW mnoży elementy dwu wektorów, w razie potrzeby transponując drugi

try
    C = A.*B;
catch
    C = A.*B';
end
```

```
>> a = 1:3, b=a'
a =
     1     2     3
b =
     1
     2
     3
>> mnozenieWektorow(a,b)
ans =
     1     4     9
```

Dziś najważniejsze było to...

- Programowanie w Matlabie jest podobne do programowania w C/C++
  - nie uczymy się programować od nowa!
    - przenosimy umiejętności z C/C++ na Matlaba
- Podstawową formą programu jest funkcja, a nie skrypt

A za 2 tygodnie...

- Typy danych – dokończenie
  - patrz slajdy do wykładu 1.5
- Optymalizacja obliczeń
- Grafika w Matlabie