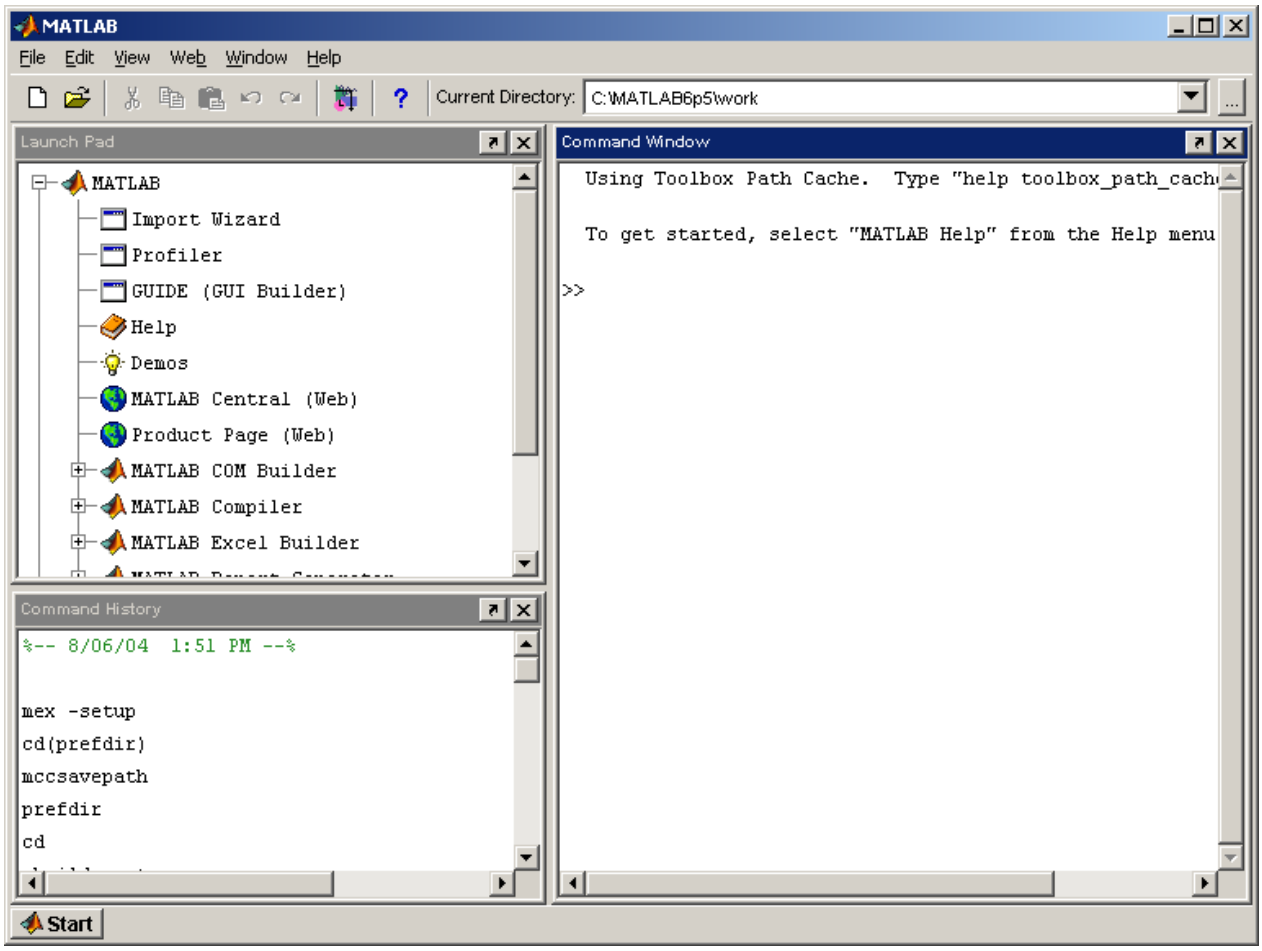
Podstawy MATLABA

MATLAB jest zintegrowanym środowiskiem obliczeniowo-programistycznym dla zastosowań inżynierskich. Umożliwia zarówno obliczenia i ich wizualizację oraz programowanie we własnym języku wysokiego poziomu. Podstawowe zastosowania MATLABA zawierają:

- obliczenia matematyczne
- budowanie algorytmów
- zbieranie danych
- modelowanie, symulacje i wizualizacje
- analiza, eksploracja oraz wizualizacja danych
- grafika dla zastosowań naukowych i inżynierskich
- budowa aplikacji wraz z GUI

1. Budowa środowiska

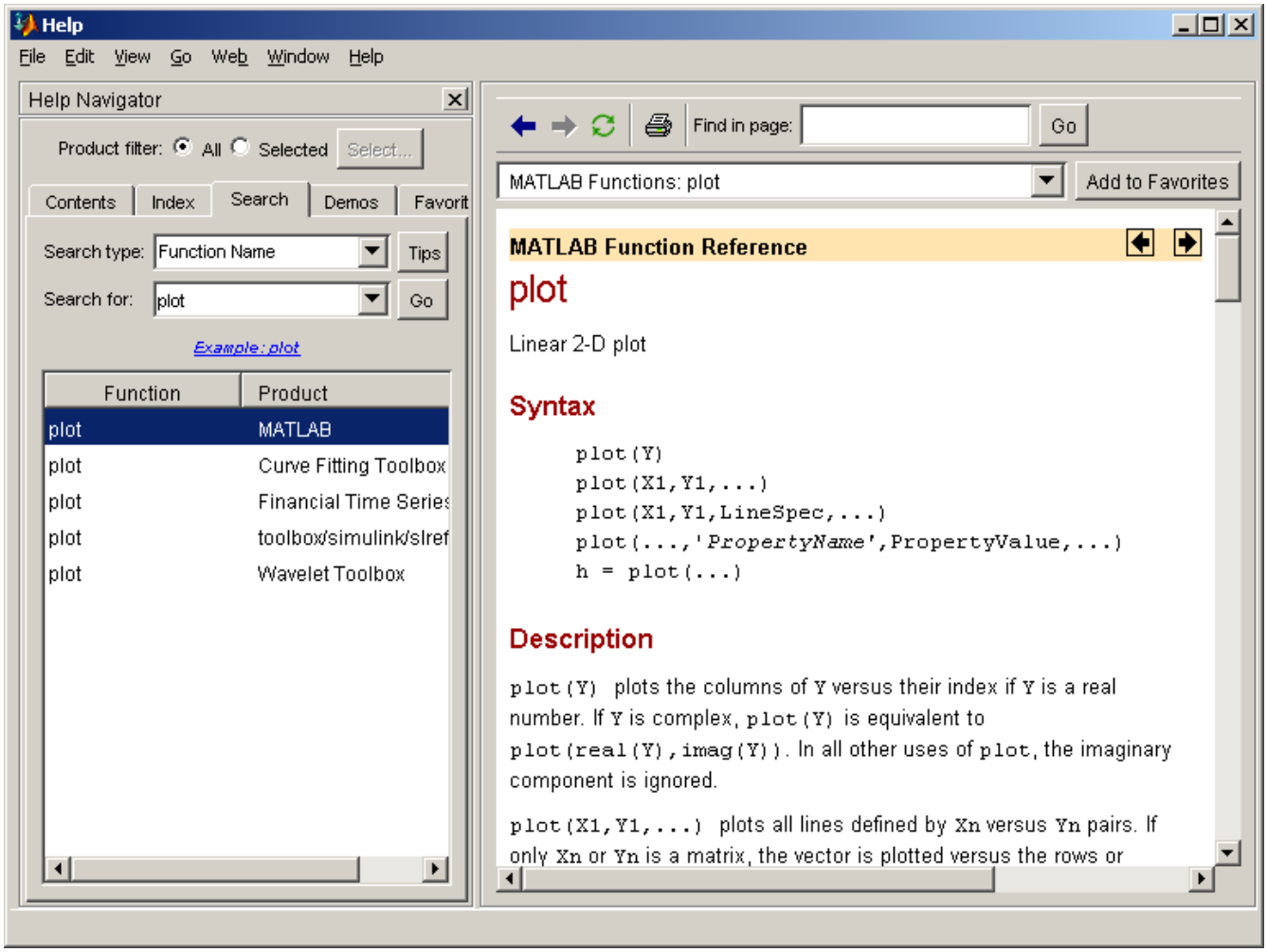


Rys.1. Typowy wygląd aplikacji.

Podstawowym oknem Matlaba jest **okno poleceń** (*Command Window*), w którym wpisuje się funkcje i komendy Matlaba. W zależności od ustawień w menu *View* mogą być widoczne także inne okna, np. okno historii (*Command History*), w którym pojawiają się wpisane wcześniej w *Command Window* polecenia lub okno *Launch Pad*, które zapewnia szybki dostęp do dokumentacji, demonstracji i innych narzędzi. Wygląd aplikacji można dostosowywać do swych własnych potrzeb.

1.1. System pomocy i dokumentacja

Wszystkie funkcje i narzędzia Matlaba są wyczerpująco opisane w dokumentacji, która jest dostępna offline (jeśli jest zainstalowana w systemie) lub online na stronie producenta: http://www.mathworks.com



Rys.2. Okno pomocy.

Pomoc na temat wybranej funkcji Matlaba można uzyskać dwojako. Najprościej jest wpisać w oknie poleceń komendę *help*, po której następuje nazwa interesującej nas funkcji, np. wpisanie:

```
help plot
```

spowoduje wyświetlenie informacji na temat funkcji *plot*, służącej do rysowania wykresów. Innym sposobem jest otwarcie okna pomocy (menu *help*), i w zakładce *search* wpisanie nazwy poszukiwanej funkcji (patrz rys. 2).

Inne przydatne funkcje:
Funkcja *which* podaje lokalizację poszukiwanej funkcji, np. wpisanie:

```
which mean
```

powoduje wyświetlenie ścieżki dostępu do funkcji *mean*, służącej do obliczania wartości średniej:

```
C:\MATLAB6p5\toolbox\matlab\datafun\mean.m
```
W tym wypadku kod źródłowy tej funkcji znajduje się w pliku mean.m w podanej lokalizacji. Jest to plik tekstowy, więc można go otworzyć i podglądnąć w jaki sposób funkcja została napisana przez programistów firmy Mathworks. Taki plik można także zmodyfikować, dostosowując go do własnych potrzeb.

Innym sposobem dotarcia do kodu źródłowego danej funkcji Matlaba jest wykorzystanie funkcji *type*, np. wpisanie w oknie poleceń:
```
type mean
```
spowoduje wyświetlenie kodu źródłowego funkcji *mean* w oknie poleceń.

2. Podstawowe komendy

2.1. Katalog roboczy

Zawsze przed rozpoczęciem pracy w Matlabie należy ustawić ścieżkę dostępu do katalogu roboczego, w którym będziemy przechowywać swoje pliki. Domyślnym katalogiem roboczym jest zazwyczaj:
C:\MATLAB6p5\work
Ponieważ nie zawsze ten katalog jest dostępny do zapisu, za pomocą funkcji *cd* należy podać katalog do którego mamy dostęp, np.:
```
cd f:\praca
```
czyli dysk *f*, katalog *praca*. Taki katalog trzeba oczywiście wcześniej utworzyć.
Inne wykorzystanie funkcji *cd*:

`cd`	- (bez argumentów) wyświetla aktualny katalog
`cd \`	- przejście do katalogu głównego
`cd ..`	- przejście o jeden poziom wyżej

2.2. Wpisywanie zmiennych

Wpisywanie zmiennych liczbowych (skalarów) jest bardzo proste, np.:
```
x = 5
a = 2.45
```
Na końcu wyrażenia można postawić średnik, aby wynik operacji nie był wyświetlany na ekranie:
```
x = 5;
a = 2.45;
```
Wektory wpisuje się w nawiasach kwadratowych, np.:
```
y = [1 2 3 4]
```
lub
```
y = [1, 2, 3, 4]
```
tworzy wektor wierszowy, natomiast:
```
z = [1; 2; 3; 4]
```
tworzy wektor kolumnowy.

Wszystkie wpisane zmienne są przechowywane przestrzeni roboczej (czyli w pamięci) dopóki nie zostaną z niej usunięte. Aby sprawdzić jakie zmienne aktualnie znajdują się w przestrzeni roboczej należy użyć funkcji *who*:
```
who
```
W wyniku otrzymamy informację:
```
Your variables are:
```
(twoje zmienne to)
```
a  x  y  z
```
Bardziej rozbudowaną funkcją *who* jest *whos*:
```
whos
```
W wyniku otrzymamy informację:

```
  Name      Size            Bytes  Class
  a         1x1                 8  double array
  x         1x1                 8  double array
  y         1x4                32  double array
  z         4x1                32  double array
Grand total is 10 elements using 80 bytes
```

Jak widać każda zmienna w Matlabie jest obiektem typu *double array* (macierz liczb zmiennoprzecinkowych o podwójnej precyzji). Nawet skalar jest traktowany jako macierz o wymiarach 1x1.

2.3. Rozmiar zmiennych

Do sprawdzania rozmiaru zmiennych służą dwie funkcje: *size* (rozmiar) oraz *length* (długość), np.:

```
size(y)
```

W wyniku otrzymamy informację:

```
ans =
     1     4
```

co oznacza, że zmienna *y* jest macierzą o rozmiarze 1x4, czyli 1 wiersz i 4 kolumny.
Natomiast funkcja *length* podaje dłuższy z wymiarów, np.:

```
length(y)
ans =
     4
```

„ans" jest skrótem od „answer" (odpowiedź).

2.4. Zapisywanie i odczytywanie zmiennych

Do zapisywania zmiennych na dysku (do tej pory istniały tylko w pamięci komputera) służy funkcja save, np.:

```
save nazwa_pliku
```

zapisuje wszystkie zmienne z przestrzeni roboczej do pliku binarnego nazwa_pliku.mat. „mat" jest domyślnym rozszerzeniem plików z danymi w Matlabie.

Zapisz wszystkie zmienne z przestrzeni roboczej do pliku, następnie znajdź ten plik w swoim katalogu roboczym (jaki jest katalog roboczy możesz sprawdzić poleceniem *cd*) i spróbuj go otworzyć w Notatniku. Co zawiera?

Inne wywołania funkcji *save*:

```
save nazwa_pliku x          - zapisuje tylko zmienną x
save nazwa_pliku x y z      - zapisuje zmienne x, y i z.
save nazwa_pliku x -ascii   - zapisuje zmienną x w formacie ascii
```

Zapisz wszystkie zmienne z przestrzeni roboczej do pliku w formacie ascii (tekstowym) i spróbuj go otworzyć w Notatniku. Co zawiera?

Operacja odwrotna, czyli wczytywanie pliku z dysku wykonuje się za pomocą funkcji *load*:

```
load nazwa_pliku        - ładuje wszystkie zmienne z pliku nazwa_pliku
load nazwa_pliku x      - ładuje tylko zmienną x z pliku nazwa_pliku
```

Do usuwanie zmiennych z przestrzeni roboczej służy funkcja *clear*:

```
clear          - usuwa wszystkie zmienne z przestrzeni roboczej
clear x y      - usuwa tylko zmienne x i y
```

3. Operacje na macierzach

Macierz jest podstawowym obiektem Matlaba. Matlab jest językiem programowania zoptymalizowanym pod względem wykonywania operacji macierzowych.

3.1. Tworzenie macierzy

Macierz tworzymy za pomocą nawiasów kwadratowych. Elementy macierzy wpisujemy po kolei wiersz po wierszu. Elementy w wierszu (czyli poszczególne kolumny) są oddzielone spacją lub przecinkiem, natomiast przejście do następnego wiersza symbolizuje średnik, np.:

```
A=[1 2;3 4]
A =
     1     2
     3     4
B=[8 7;6 5]
B =
     8     7
     6     5
C = [1 2; 4 5; 6 7]
C =
     1     2
     4     5
     6     7
```

3.2. Dodawanie macierzy

```
A+B
ans =
     9     9
     9     9
```

3.3. Odejmowanie macierzy

```
A-B
ans =
    -7    -5
    -3    -1
```

3.4. Mnożenie macierzowe

Uwaga: przy mnożeniu A*B liczba kolumn macierzy A musi być równa liczbie wierszy macierzy B.

```
A*B
ans =
    20    17
    48    41
```

Jeżeli powyższy warunek nie jest spełniony generowany jest komunikat o błędzie, np.:

```
A*C
??? Error using ==> *
Inner matrix dimensions must agree
```
(wymiary macierzy muszą się zgadzać)

3.5. Mnożenie tablicowe (z kropką)

Mnożenie tablicowe różni się zasadniczo od macierzowego, polega ono na mnożeniu odpowiadających sobie elementów (o takich samych indeksach), np.:

```
A.*B
ans =
     8    14
    18    20
```
Uwaga: W tym przypadku wymiary obu macierzy muszą być jednakowe!

3.6. Transpozycja macierzy (apostrof)
```
A'
ans =
     1     3
     2     4
```

3.7. Wyznacznik macierzy
```
det(A)
ans =
    -2
```

3.8. Wymiar macierzy
```
rank(A)
ans =
     2
```

3.9. Odwrotność macierzy
```
inv(A)
ans =
   -2.0000    1.0000
    1.5000   -0.5000
```

3.10. Potęgowanie macierzowe
```
A^2
ans =
     7    10
    15    22
```

3.11. Potęgowanie tablicowe (z kropką)
Podobnie jak w przypadku mnożenia tablicowego, do potęgi podnoszony jest każdy element macierzy, np.:
```
A.^2
ans =
     1     4
     9    16
```

3.12. Odwoływanie się do elementów macierzy
Znajomość operacji na poszczególnych elementach macierzy jest kluczowa dla zrozumienia języka Matlab. Załóżmy, że mamy macierz o wymiarach 3x3, np.:
```
E=[1 2 3;4 5 6;7 8 9]
E =
     1     2     3
     4     5     6
     7     8     9
```
Element o indeksie (1,1) czyli na przecięciu pierwszego wiersza i pierwszej kolumny:

```
E(1,1)
ans =
     1
```
Element o indeksie (2,3) czyli na przecięciu drugiego wiersza i trzeciej kolumny:
```
E(2,3)
ans =
     6
```
Podmacierz składająca się z trzeciego wiersza i kolumny od pierwszej do drugiej:
```
E(3,1:2)
ans =
     7     8
```
Podmacierz składająca się z drugiego wiersza i wszystkich kolumn:
```
E(2,:)
ans =
     4     5     6
```
Podmacierz składająca się ze wszystkich wierszy i drugiej kolumny:
```
E(:,2)
ans =
     2
     5
     8
```
Podmacierz składająca się z wiersza od pierwszego do drugiego i wszystkich kolumn:
```
E(1:2,:)
ans =
     1     2     3
     4     5     6
```
Zamiana macierzy na wektor kolumnowy:
```
E(:)
ans =
     1
     4
     7
     2
     5
     8
     3
     6
     9
```

Przydatna sztuczka:
Jak uzyskać wektor z połączonych wierszy? Należy wykonać dwie poniższe operacje:
`F=E'` najpierw transpozycja macierzy E i zapisanie wyniku do macierzy F
`F(:)'` następnie zamiana macierzy F na wektor kolumnowy i transpozycja wyniku
```
ans =
     1     2     3     4     5     6     7     8     9
```

Operator : (dwukropek)
Inne zastosowania dwukropka to tworzenie wektorów o elementach oddalonych od siebie o taką samą wartość, np. wpisanie polecenia:
```
1:10
```
daje w wyniku wektor liczb od 1 do 10:

```
ans =
     1     2     3     4     5     6     7     8     9    10
```
Domyślna inkrementacja wynosi 1 ale można podać dowolną, np.:
```
1:2:10
```
daje w wyniku wektor liczb od 1 do 10 z krokiem 2:
```
ans =
     1     3     5     7     9
```
Można zastosować inkrementację ujemną, np.:
```
10:-2:1
ans =
    10     8     6     4     2
```

4. Wykorzystanie rachunku macierzowego do rozwiązywania układów równań

4.1. Przykład z macierzą kwadratową

Rozwiązać układ równań:

$$\begin{cases} 3x_1 + 2x_2 + x_3 = 5 \\ 2x_1 + 3x_2 + x_3 = 1 \\ 2x_1 + x_2 + 3x_3 = 11 \end{cases}$$

W zapisie macierzowym równanie to można przedstawić w następującej formie:
A * **x** = **b**
gdzie **A** jest macierzą a **x** oraz **b** są wektorami. Najpierw zatem należy utworzyć A i b:
```
A=[3 2 1;2 3 1;2 1 3]
A =
     3     2     1
     2     3     1
     2     1     3
b=[5;1;11]
b =
     5
     1
    11
```
Rozwiązaniem układu równań liniowych o postaci:
A * **x** = **b**
jest:
x = $\mathbf{A}^{-1}$ * **b**
W tym przypadku A jest kwadratowa, więc rozwiązanie można uzyskać z zależności:
```
x = inv(A)*b
```
co daje w wyniku:
```
x =
    2.0000
   -2.0000
    3.0000
```

4.2. Przykład z macierzą prostokątną

Rozwiązać układ równań:

$$\begin{cases} 2x_1 + 5x_2 = 7 \\ 3x_1 - x_2 = 2 \\ 4x_1 + 6x_2 = 10 \end{cases}$$

Postępujemy podobnie jak w poprzednim przykładzie

```
A=[2 5;3 -1;4 6]
A =
     2     5
     3    -1
     4     6
b=[7;2;10]
b =
     7
     2
    10
```

Jednak usiłując odwrócić macierz A dostajemy w typ przypadku komunikat błędu:

```
inv(A)*b
??? Error using ==> inv
Matrix must be square.
```

Nie można tu użyć funkcji odwracania macierzy, bo A nie jest prostokątna.
Wyjściem z sytuacji jest zastosowanie operatora dzielenia lewostronnego \ używanego m.in. do rozwiązywania układu równań liniowych o postaci:
A * x = b
rozwiązanie otrzymujemy z zależności:
x = A \ b
Matlab automatycznie dokonuje wyboru optymalnej metody rozwiązania w zależności od struktury macierzy A. W wyniku otrzymujemy:

```
x=A\b
x =
    1.0000
    1.0000
```

5. Clown – inny przykład wykorzystania macierzy

Macierze mogą służyć nie tylko do obliczeń numerycznych. Oto przykład.
Wczytaj dane:

```
load clown
```

Sprawdź co zostało wczytane:

```
whos
  Name          Size       Bytes    Class

  X           200x320      512000  double array
  caption       2x1        4       char array
  map          81x3        1944    double array
```

Jak widać mamy tu między innymi macierz X o wymiarach 200x320. Podglądnij kawałek tej macierzy o wymiarach 10x10:

```
X(1:10,1:10)
```

Następnie zastosuj funkcję *image* (do wyświetlania obrazów) oraz *colormap* (do ustawiania mapy kolorów):

```
image(X), colormap(map)
```

Zastanów się co oznaczają wartości w macierzy X.