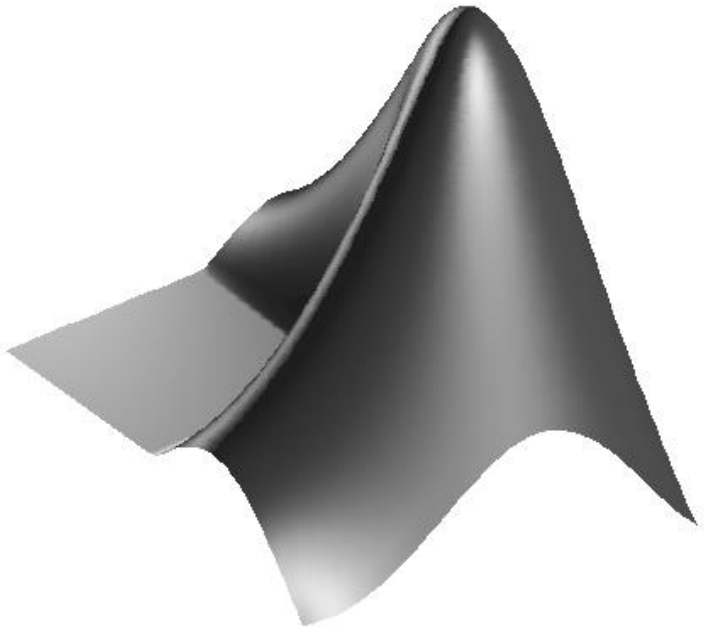


KURS MATLAB I

Rok 2024/2025, semestr letni

Uniwersytet Warszawski
Wydział Fizyki

Krzysztof Markowicz,

Na podstawie i modyfikacji skryptu autorstwa
Ryszarda Buczyńskiego i Rafała Kasztelanica

Spis Treści

KURS MATLAB I

Rok 2024/2025 semestr letni, wymiar 30h

Prowadzący:

Prof. dr hab. Krzysztof Markowicz, kmark@igf.fuw.edu.pl,
www.igf.fuw.edu.pl/~kmark/stacja/wyklady/MATLAB

Zakład Fizyki Atmosfery, Instytut Geofizyki, Wydz. Fizyki UW, Pasteura 5, pok. B4.47
Zajęcia odbywają się w budynku przy ul. Pasteura 5 w sali 1.30/1.29 lub zdalnie:

Oprogramowanie:

Matlab, *The MathWorks, Inc.;* wersja 2019 lub nowsza; platforma UNIX/LINUX.

Charakterystyka kursu:

Poziom podstawowy, wymagana znajomość podstawowych pojęć matematycznych z zakresu algebry, analizy matematycznej i prawdopodobieństwa, znajomość programowania nie jest konieczna, ale mile widziana.

Forma zaliczenia:

Do zaliczenia Kursu na ocenę dostateczną wymagane jest zaliczenie wszystkich ćwiczeń-laboratoriów. Ocena końcowa wystawiana jest przez prowadzącego na podstawie osiągniętej sprawności i postępów w posługiwaniu się środowiskiem obliczeniowym MATLAB, kreatywności studenta oraz kolokwium na ostatnich zajęciach. Obowiązkowe jest uzyskanie certyfikatu z kursu Matlaba online:
https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted
Obecność na wszystkich zajęciach jest obowiązkowa, dopuszczana jest jedna nieobecność, przy większej ilości wymagane jest zwolnienie lekarskie. Nieobecność nie zwalnia studenta z zaliczenia poszczególnych zadań.

Spis omawianej problematyki:

1. Opis środowiska Matlaba
2. Operacje algebraiczne na wektorach i macierzach
3. Wizualizacja danych – Wykresy dwuwymiarowe
4. Wizualizacja danych – Wykresy trójwymiarowe
5. Podstawy programowania: skrypty i funkcje
6. Instrukcje w Matlabie
7. Inne przydatne funkcje
8. Rozwiązywanie równań nieliniowych
9. Rozwiązywanie układów równań liniowych
10. Interpolacja i aproksymacja funkcji
11. Podstawy statystyki w Matlabie

Literatura:

1. Matlab: Intro, Demo, manual online.
2. A. Zalewski R. Cegieła, Matlab – *Obliczenia numeryczne i ich zastosowania*, Wyd. Nakom, Poznań 1996.
3. B. Mrożek, Z. Mrożek, Matlab uniwersalne środowisko do obliczeń naukowo-technicznych, Wyd. PLJ, Warszawa 1996
4. B. Mrozek, Z. Mrozek, Matlab 6 – poradnik użytkownika
5. Waldemar Sradomski, MATLAB. Praktyczny podręcznik modelowania, Helion, Gliwice, 2015
6. Pratap Rudra, MATLAB 7 dla naukowców i inżynierów, 2010
7. Pełna dokumentacja, forum dyskusyjne, i wiele innych rzeczy www.mathworks.com

TEMATY

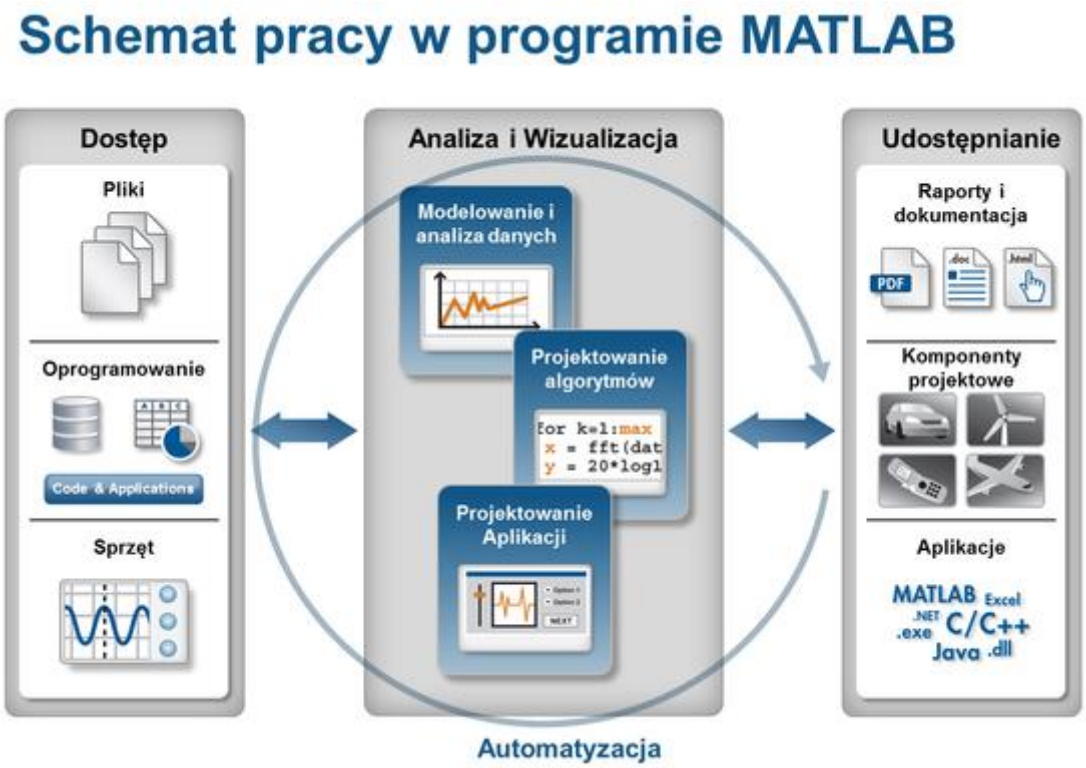
OPIS ŚRODOWISKA MATLABA

Temat 1
Matlab: rys historyczny, przeznaczenie oprogramowania i opis pakietu

MATLAB – program komputerowy będący interaktywnym środowiskiem do wykonywania obliczeń naukowych i inżynierskich oraz do tworzenia symulacji komputerowych.

Nazwa programu pochodzi od angielskich słów *MATrix LABoratory*, gdyż początkowo program ten był przeznaczony do numerycznych obliczeń macierzowych. Obecnie program ten potrafi znacznie więcej, cechuje go duża liczba funkcji bibliotecznych oraz duże możliwości rozbudowy przez użytkownika za pomocą pisania własnych funkcji. Posiada on swój język programowania, co umożliwia pisanie w pełni funkcjonalnych programów działających w środowisku Matlaba.

W zakresie grafiki MATLAB umożliwia rysowanie dwu i trójwymiarowych wykresów funkcji oraz wizualizację wyników obliczeń w postaci rysunków statycznych i animacji. Możliwe jest pobieranie danych pomiarowych i sterowanie urządzeniami zewnętrznymi przez porty w celu. Istnieją alternatywne odpowiedniki tegoż programu rozprowadzane na licencjach FLOSS, takie jak Scilab czy Octave, jednak nie są tak rozbudowane jak MATLAB



http://www.ont.com.pl/produkty/lista-produktow/matlab/

Prapoczątki programu MATLAB sięgają lat siedemdziesiątych, gdy w USA na zlecenie National Science Foundation powstały biblioteki języka Fortran do obliczeń macierzowych: Linpack i Eispack. Jeden z autorów tych bibliotek, Cleve Moler prowadził zajęcia z algebry

liniowej na Uniwersytecie stanu Nowy Meksyk. Chcąc ułatwić życie swoim studentom napisał on w 1980 r. program, który umożliwiał korzystanie z tych bibliotek bez potrzeby programowania w Fortranie. Program ten napisany (także w Fortranie) w formie prostego interaktywnego języka poleceń i rozprowadzany na zasadach public domain był pierwowzorem programu MATLAB. W 1983 C. Moler oraz S. Bangert i J. Little (inżynier z Uniwersytetu Stanford) postanowili rozwinąć powyższy projekt – zastąpili Fortran językiem C i dodali zintegrowaną grafikę. Założyli oni firmę The MathWorks Inc., która do dziś zajmuje się rozwojem i sprzedażą pakietu Matlab. W 1985 roku pojawiła się pierwsza wersja programu (wikipedia).

Najważniejsze rodzaje plików
M-pliki (*.m)
W celu zautomatyzowania pewną liczbę poleceń z wiersza poleceń możemy zapisać do pliku z rozszerzeniem *.m (stąd nazwa m-plik) i tę listę poleceń uruchomić jednym poleceniem – mówimy wówczas o m-pliku skryptowym.

Mex-pliki
Program napisany w języku C lub Fortran możemy skompilować poleceniem mex. Wynikiem kompilacji jest otrzymanie pliku dynamicznie ładowanej biblioteki współdzielonej (w Microsoft Windows są to pliki *.dll nazywanego mex-plikiem) (skrót od *Matlab EXecutable*). Mex-plik można uruchomić z wiersza poleceń w oknie programu MATLAB tak jak zwykły m-plik.

Mat-pliki (*.mat)
Tymczasowe lub końcowe wyniki obliczeń możemy zapisywać do pliku tekstowego ASCII o dowolnej nazwie lub do pliku binarnego z rozszerzeniem *.mat – wówczas wszystkie liczby (całkowite i zmiennoprzecinkowe) zapisywane są w formacie zmiennoprzecinkowym z podwójną precyzją.

Pliki fig (*.fig)
Wykresy i inne formy graficzne wygenerowane w programie MATLAB można zapisać do wybranego formatu graficznego lub do pliku binarnego z rozszerzeniem *.fig. Zaletą tej drugiej formy zapisu jest możliwość późniejszej modyfikacji zapisanego obiektu w programie. Zapisu do formatu fig możemy dokonać za pomocą wybrania odpowiedniej opcji z menu (*save as*) lub za pomocą polecenia *saveas*. Wczytania pliku *.fig do programu MATLAB dokonuje się poleceniem *open* lub *openfig*.

Pliki Live script (*.mlx)

Live Scripty w MATLAB-ie to interaktywne skrypty, które umożliwiają łączenie kodu, wyników obliczeń, wykresów oraz formatowanego tekstu w jednym dokumencie. Dzięki temu można łatwo tworzyć dynamiczne raporty, dokumentacje oraz przejrzyste analizy danych. Są szczególnie przydatne w nauczaniu, eksploracji danych i współpracy, ponieważ pozwalają na natychmiastową wizualizację wyników oraz użycie elementów interaktywnych, takich jak suwaki czy pola wejściowe.

Pliki aplikacji (*.mlapp) App Designer

Pliki aplikacji MATLAB App Designer to specjalny format przechowujący interfejsy graficzne (GUI) stworzone w środowisku MATLAB. Zawierają zarówno kod źródłowy

aplikacji, jak i układ wizualny z elementami interaktywnymi, takimi jak przyciski, pola tekstowe czy wykresy. Dzięki temu umożliwiają tworzenie intuicyjnych narzędzi do analizy danych, symulacji czy wizualizacji wyników bez potrzeby pisania kodu obsługującego GUI od podstaw.

Język programowania pakietu MATLAB jest pełnoprawnym językiem programowania wysokiego poziomu, o składni wzorowanej na języku C. Pozwala on na używanie funkcji i struktur, oraz umożliwia pisanie programów zorientowanych obiektowo. Tak jak wszystkie współczesne języki programowania wysokiego poziomu posiada on instrukcje sterujące takie jak: if, for, while, switch. Rezygnacja z trójargumentowej pętli for na rzecz tzw. notacji dwukropkowej skraca kod źródłowy, a więc i czas pisania.

Toolbox'y (z ang. *toolboxes*) to zbiór dodatkowych bibliotek (m-plików) do rozwiązywania specjalistycznych problemów z określonych dziedzin (automatyka, elektronika, telekomunikacja, matematyka etc.). Biblioteki te rozszerzają możliwości programu MATLAB.

Wybrane toolbox'y

- Fuzzy Logic Toolbox – środowisko do projektowania i diagnostyki inteligentnych układów sterowania wykorzystujących metody logiki rozmytej i uczenie adaptacyjne.

- Image Processing Toolbox – programowe narzędzia do przetwarzania obrazów.

- Mapping Toolbox – przeznaczony do analizy informacji geograficznych i wyświetlania map, z możliwością dostępu do zewnętrznych źródeł geograficznych.

- Neural Network Toolbox – zbiór funkcji do projektowania i symulacji sieci neuronowych.

- Symbolic Math Toolbox – zestaw funkcji do obliczeń symbolicznych – rozszerza możliwości programu MATLAB o możliwość wykonywania obliczeń symbolicznych.

- Partial Differential Equation Toolbox – zestaw funkcji do numerycznego rozwiązywania równań różniczkowych cząstkowych metodą elementów skończonych.

- Simulink– pakiet służący do modelowania, symulacji i analizy układów dynamicznych. Simulink dostarcza także graficzny interfejs użytkownika umożliwiający konstruowanie modeli w postaci diagramów blokowych.

- Spline Toolbox – zestaw bibliotek do aproksymacji i interpolacji funkcjami sklejanymi.

- Wavelet Toolbox – biblioteka do analizy falowej sygnałów.

- Optimization Toolbox – zestaw funkcji służących do wyznaczania minimum czy maksimum obiektów z uwzględnieniem szeregu warunków

- Instrument Control Toolbox – zestaw bibliotek do komunikacji z urządzeniami zewnętrznymi

- Parallel Computing Toolbox- umożliwia wykonywanie obliczeń równoległych na procesorach wielordzeniowych lub kastrach komputerowych.

MatLAB online
To wersja MatLAB'a, która może być używana z poziomu przeglądarki po wcześniejszym zarejestrowaniu na stronie https://www.mathworks.com. Dostęp do tej wersji matlaba: https://matlab.mathworks.com/
Każdy użytkownik ma dostępną przestrzeń dyskową na tzw. Matlab drive, w której może przechowywać dane oraz skrypty. Bezpłatny czas pracy w środowisku MATLAB wynosi 30h na miesiąc.

Kursy interaktywne
Mathworks poprzez produkt MatLAB Onramp oferuje szereg interaktywnych kursów obejmujących podstawy MatLAB'a oraz wybrane zagadnienia zaawansowane.
https://matlabacademy.mathworks.com/

Temat 2
Operowanie Matlabem w środowisku Linux i Windows

Okna: workspace, directory, history, array editor, …

Temat 3
Różnice między wersjami Matlaba – funkcja *ver*

```
>> ver % podaje numer wersji Matlaba oraz numery zainstalowanych dodatków
```

Temat 4
Zapoznanie się z narzędziami wprowadzającymi Matlaba – funkcje *demo, peaks, bench*

```
>> demo % wyświetla dostępne przykłady
>> peaks % przykładowa funkcja 2 zmiennych
>> bench % sprawdzenie szybkości pracy Matlaba – benchmark
```

Temat 5
Poszukiwanie znaczeń funkcji i skryptów – funkcja *help*

```
>> help % wypisuje linki do wszystkich plików pomocy
>> help plot % wypisuje pomoc dotyczącą funkcji plot
```

Temat 6
Szukanie za pomocą słów kluczowy: *lookfor*

```
>> lookfor bessel % przeszukuje pliki pomocy szukając słowa kluczowego bessel
```

Temat 7
Znaczenie średnika na końcu polecenia

Średnik kończący komendę w Matlabie powoduje, że wynik działania danej komendy nie będzie wyświetlany na ekranie.

Temat 8
Symbole operatorów

=	Przypisanie wartości
[]	Tworzenie macierzy, list argumentów wyjściowych funkcji
()	Listy argumentów wejściowych funkcji, kolejność działań matematycznych
.	Kropka dziesiętna, część operatorów arytmetycznych
..	Katalog macierzysty
...	Kontynuacja polecenia jest w następnej linii
, .	Symbole separacji argumentów funkcji, indeksów, itp.
;	Koniec wiersza macierzy, koniec polecenia bez wypisywania odpowiedzi
%	Początek linii komentarza
:	Generowanie wektorów, indeksowanie macierzy
‘	Początek i koniec wprowadzania łańcuchów znakowych, transpozycja macierzy, sprzężenie macierzy
! **unix** **doc** **system**	Komenda sytemu operacyjnego Komendy systemu operacyjnego
ispc	zwraca 1 jest system operacyjny to Windowsy
isunix	zwraca 1 jest system operacyjny to UNIX

Temat 9
Zmienne specjalne i stałe

ans	**Zmienna robocza, automatycznie przyjmuje daną wartość, jeśli nie nadano jej nazwy**
computer	**Nazwa komputera, na którym działa Matlab**
eps	**Precyzja zmiennoprzecinkowa**
flops	**Licznik operacji zmiennoprzecinkowej**
i, j	**Jednostka liczby urojonej**
inf	**Nieskończoność**
NaN	**Wartość nieokreślona (zwykle oznacza wprowadzenie wartości nieliczbowej jako argumentu funkcji matematycznej)**
nargin	**Liczba argumentów wejściowych funkcji**
nargout	**Liczba argumentów wyjściowych funkcji**
pi	**3.1415926....**
realmax	**Największa dostępna liczba rzeczywista**
realmin	**Najmniejsza dostępna liczba rzeczywista**

Temat 10
Podstawowe funkcje matematyczne

abs	Wartość bezwzględna, moduł liczby zespolonej, wektor wartości znaków łańcucha
acos, acosh	Arcus cosinus, arcus cosinus hiperboliczny
acot, acoth	Arcus cotangens, ...
acsc, acsch	Arcus cosecans, ...
angle	Kąt fazowy dla liczby zaspolonej w radzianach
asec, asech	Arcus secans, ...

asin, asinh	Arcus sinus, ...
atan, atanh	Arcus tangens, ...
atan2	Arcus tangens, wynik w przedziale [-π, π]
ceil	Zaokrąglenie w górę, sufit
conj	Liczba sprzężona do liczby
cos, cosh	Cosinus, ...
cot, coth	Cotangens, ...
csc, csch	Cosecans, ...
exp	e do potęgi argumentu
fix	Zaokrąglenie w kierunku zera
floor	Zaokrąglenie w dół, podłoga
gcd	Największy wspólny podzielnik
imag	Część urojona liczby zespolonej
lcm	Najmniejsza wspólna wielokrotność
log	Logarytm naturalny argumentu
log10	Logarytm dziesiętny argumentu
real	Część rzeczywista liczby zespolonej
rem	Reszta z dzielenia
round	Zaokrąglenie do najbliższej liczby całkowitej
sec, sech	Secans, ...
sign	Znak funkcji
sin, sinh	Sinus, ...
sqrt	Pierwiastek kwadratowy
tan, tanh	Tangens, ...

Przykład:

```
>> abs(5+3i) % wyświetla wartość bezwzględną liczby zespolonej
>> complex(a,b)   % definicja liczby zespolonej o części rzeczywistej a i urojonej b
```

Temat 11

Wprowadzanie zmiennych różnych typów

```
>> a=´łańcuch wprowadzany´; % zmienna łańcuchowa
>> z=3+2i; zmienna zespolona (część urojoną oznaczamy literą i lub j
```

Temat 12

Wprowadzanie precyzji wyświetlanych wyników – funkcja format

Do ustalenia precyzji wyświetlania wyników służy funkcja FORMAT.

UWAGA: Wszystkie obliczenia w MATLABie wykonywane są w podwójnej precyzji.

Polecenie	**Wartość**	**Opis**
Format short	3.1416	5 cyfr, reprezentacja stałoprzecinkowa
Format long	3.14159265358979	15 cyfr, reprezentacja stałoprzecinkowa
Format shortE	3.1416e+000	5 cyfr, reprezentacja zmiennoprzecinkowa
Format longE	3.141592653589793e+000	15 cyfr, reprezentacja zmiennoprzecinkowa
Format shortG	3.1416	5 cyfr, reprezentacja stało- lub zmiennoprzecinkowa
Format longG	3.14159265358979	15 cyfr, reprezentacja stało- lub zmiennoprzecinkowa
Format hex	400921fb54442d18	Liczba w układzie szesnastkowym
Format bank	3.14	2 liczby dziesiętne, np. złoty i grosze
Format rat	355/113	Przybliżona wartość liczby w postaci ułamka
Format +	+	Informacja o znaku liczby

Temat 13

Informacja i usuwanie zmiennych z przestrzeni roboczej – funkcje *who, whos, clear*

```
>> who % informacja o dostępnych zmiennych, same nazwy
>> whos % pełna informacja o dostępnych zmiennych
>> clear a % usunięcie z przestrzeni roboczej zmiennej a
>> clear all % usunięcie wszystkich zmiennych
```

Temat 14
Zmienne losowe w Matlabie

```
>> rand % rozkład równomierny w przedziale (0,1)
>> randn % rozkład normalny o odchyleniu standardowym 1 i wariancji 1
```

Po każdym uruchomieniu Matlaba funkcja rand startuje od tych samych wartości. Aby zacząć od innej wartości należy wywołać funkcję rand w następujący sposób:

```
>> rand('state',sum(100*clock)) % wartość początkowa na podstawie wskazań zegara
```

Temat 15
Informacje o operatorach – *help ops*

*****	mnożenie macierzy
/	dzielenie macierzy (lewej przez prawą)
\	dzielenie macierzy (prawej przez lewą)
^	podnoszenie do potęgi
‘	sprzężenie macierzy
.*	mnożenie tablicowe
./	dzielenie tablicowe (lewej przez prawą)
.	dzielenie tablicowe (prawej przez lewą)
.’	transpozycja macierzy
.^	tablicowe podnoszenie do potęgi

Temat 16
Operatory relacji

==	Relacja równości
~=	Relacja nierówności
<	Relacja mniejszości
>	Relacja większości
<=	Relacja mniejsze-równe
>=	Relacja większe-równe

Temat 17
Operatory logiczne

& (and)	Logiczne i
| (or)	Logiczne lub
~ (not)	Logiczne nie
Xor	Operacja exclusive or
any	Operacja logiczna - jeśli jakiś
all	Operacj alogiczna - wszystkie

Temat 18
Długie linie

```
>> x=1 + 1/2 + 1/3 + 1/4 + 1/5 + …
    1/6 + 1/7 + 1/8 + 1/9 + 1/ 10;
```

Temat 19
Kilka instrukcji w jednej linii

Poszczególne instrukcje oddzielamy przecinkiem.

Przykład:

```
>> x=2;, y=4;
```

Temat 20
Czyszczenie okna komend – funkcja *clc*

Temat 21
Wyprowadzanie na ekran tekstów – funkcja *disp*

```
>> disp('WYNIK: '), disp(2+2) % wyświetli WYNIK: 4
```

Dla znających składnię języka C wygodna może być w użyciu funkcja *fprintf()*

Temat 22
Wprowadzanie danych – funkcja *input*

Jeśli chcemy, aby użytkownik wprowadził jakąś zmienną stosujemy funkcję *input*

Przykład:

```
>> x = input('Podaj wartość: ')
```

```
Podaj wartość: 4
x = 4
```

Temat 23
Zapisywanie i wczytywanie zmiennych z pliku – funkcje *save, load*

Dokładny opis funkcji – help save, help load.

Wybrane polecenia:

```
>> save NazwaPliku x % zapisuje zmienną x w pliku NazwaPliku.mat
>> save NazwaPliku x -ascii % zapisuje zmienną x w pliku tekstowym NazwaPliku.mat
>> save NazwaPliku % zapisuje wszystkie zmienne w pliku NazwaPliku.mat
>> load NazwaPliku % wczytuje wszystkie zmienne z pliku NazwaPliku.mat
```

OPERACJE ALGEBRAICZNE NA WEKTORACH I MACIERZACH

Temat 24
Generacja macierzy za pomocą funkcji specjalnych Matlaba

eye	Macierz jednostkowa – z jedynkami na przekątnej
linspace	Wektor o wartościach rozłożonych równolegle
logspace	Wektor o wartościach rozłożonych logarytmicznie
meshgrid	Macierz dla wykresów 3D
ones	Macierz jedynek
rand	Macierz losowa o rozkładzie równomiernym
randn	Macierz losowa o rozkładzie normalnym
zeros	Macierz zer
compan	Macierz stowarzyszona
hadamard	Macierz Hadamarda
hankel	Macierz Hankela
hilb	Macierz Hilberta
invhilb	Odwrotna macierz Hilberta
magic	Kwadrat magiczny
pascal	Macierz Pascala
toeplitz	Macierz Toeplitza
vander	Macierz Vandermondea
gallery	Para małych macierzy testowych
sparse	Macierz rzadka
repmat	Tworzy macierz jako kopie mniejszej podmacierzy
unique	Usuwa z macierzy powtarzające się elementy
Kron	Iloczyn Kroneckera

Przykład:

```
>> x=ones(3); % macierz kwadratowa 3x3 z samymi jedynkami
>> y=zeros(5,2); % macierz zer o 5 wierszach i 2 kolumnach
>> z=rand(1,5); % wektor liczb losowych o 5 elementach
```

Temat 25
Generacja macierzy przy użyciu dwukropka

Przykład:

```
>> a=j:k % generuje wektor [j, j+1, ..., k-1, k]
>> a=j:i:k % generuje wektor [j, j+i,j+2i, ...,k]
```

Temat 26
Wybór elementów macierzy

Przykład:

```
>> x(:,i) % i-ta kolumna macierzy a
>> y(i,:) % i-ty wiersz macierzy y
>> z(i,a:b) % kolumny macierzy z(a) … z(b)
>> a(:) % całą macierz w postaci wektora kolumnowego
>> b(j:k) % wypisuje elementy macierzy A od elementu j do elementu k
```

Temat 27
Generacja wektorów, alokacja pamięci

Ze względu na czas wykonywania operacji dobrze jest przed przystąpieniem do obliczeń stworzyć odpowiednie macierze do przechowywania danych

Przykład:

```
>> x(5)=0; % wektor 5 elementowy wypełniony zerami
>> y(5,7)=0; % macierz 5x7 wypełniona zerami, lub y=zeros(5,7);
```

Temat 28
Znaczenie spacji, przecinka i średnika w generacji macierzy

```
>> x=[1 2 3] % wektor poziomy {1, 2, 3}, równoważne x=[1, 2, 3]
>> y=[1;2;3] % wektor pionowy {1, 2, 3}
```

Temat 29
Macierze wielowymiarowe

Przykład:

```
>> x=zeros(i,j,k); % 3-wymiarowa macierz zer o i wierszach, j kolumnach oraz k warstwach
```

Mając gotowe macierze mogę je składać w macierze o większej liczbie wymiarów. Służy do tego funkcja cat(dim,a,b,…) gdzie dim określa wzdłuż którego wymiaru dokonywane jest złożenie macierzy.

Przykład:

```
>> a=zeros(3); , b=ones(3); % dane początkowe
>> x=cat(1,a,b) % macierz 2-wymiarowa a nad b, równoważne [a; b]
>> y=cat(2,a,b) % macierz 2-wymiarowa b za a, równoważne [a b]
>> z=cat(3,a,b) % macierz 3-wymiarowa o warstwach a i b
```

Temat 30
Działania macierzowe i tablicowe

Operator	**Operacja macierzowa**	**Operacja tablicowa**
Dodawanie	+	+
Odejmowanie	-	-
Mnożenie	*	.*
Dzielenie lewostronne	\	.\
Dzielenie prawostronne	/	./
Potęgowanie	^	.^

Temat 31
Operacje na elementach wektora

max(x)	zwraca największą wartość w wektorze x. Jeśli x jest macierzą funkcja max(x) zwraca wektor gdzie kolejne elementy określają największe wartości w każdej z kolumn.
min(x)	zwraca wartość minimalną w wektorze x.
sum(x)	zwraca sumę wszystkich elementów wektora x.
prod(x)	zwraca iloczyn wszystkich elementów wektora x.
diff(x)	zwraca różnicę między kolejnymi elementami wektora x. [x(2)-x(1), x(3)-x(2), ...].

Uwaga: Jeśli x jest macierzą powyższe funkcje odnoszą się do poszczególnych kolumn macierzy x.

Temat 32
Wektoryzacja

Matlab optymalizowany jest do wykonywania działań na wektorach i macierzach. Jeśli to tylko możliwe należy dążyć do wykonywania obliczeń na wektorach lub macierzach.

Przykład:

```
% można tak – sposób iteracyjny
>> x=1:10; , y=zeros(10);
>> for i=1:10, y(i)=x(i)^2;, end

% lepiej jednak tak – sposób macierzowy
>> x=1:10;
>> y=x.^2;
```

Temat 33
Operowanie macierzami

flipdim	Wywinięcie macierzy wzdłuż danego wymiaru
fliplr	Wywinięcie macierzy w kierunku lewo-prawo
flipud	Wywinięcie macierzy w kierunku góra-dół
reshape	Zmiana rozmiaru macierzy, zmiana liczby wymiarów
rot90	Obrót macierzy o 90 stopni
squeeze	Usunięcie 1 wymiaru
tril	Macierz trójkątna dolna
triu	Macierz trójkątna górna

Temat 34
Inne przydatne funkcje

size	Podaje rozmiar macierzy
Numer	Liczba elementów w macierzy
end	Ostatni element macierzy, wektora, …
Isequal	Sprawdza czy macierze są sobie równe (funkcje typu is*)
a>0	Macierz zerojedynkowa. Jedynki tam gdzie a>0
any(a>0)	1 gdy jakiś element macierzy >0
find	Znajduje elementy spełniające dane kryterium

Przykład:

```
>> [x,y]=size(a)
>> m=numel(a)
>> b=a(5:end) % elementy wektora od 5 do końca
>> b=(a>0)
>> c=find(a>2)
```

Temat 35
Macierze rzadkie

Macierzą rzadką nazywamy macierz, której przeważająca większość elementów równa jest 0 a tylko nieliczne mają wartość znaczące. W takim przypadku ze względów pamięciowych wygodnie jest pamiętać macierz nie jako tablicę, ale poszczególne liczby wraz z adresami.

Przykład:

```
>> a= sparse([1 2 2 4 4],[3 1 4 2 4],1:5) % tworzy macierz rzadką o elementach: (2,1)->2,
        (4,2)->4, (1,3)->1, (2,4)->3, (4,4)->5
```

Aby przekształcić macierz rzadką w macierz w postaci normalnej normalną korzystamy z funkcji *full()*.

WIZUALIZACJA DANYCH
WYKRESY DWUWYMIAROWE

Temat 36
Wykresy dwuwymiarowe funkcji – funkcja *plot*

plot(X) – rysuje wektor X w funkcji indeksu, w przypadku macierzy traktuje ją jak zestaw wektorów
plot(X,Y) – wykreśla wektor Y w funkcji wektora X, Gdy X lub Y jest macierzą to wektor jest rysowany odpowiednio w funkcji kolumn lub rzędów.
plot(X,Y,S) – wykreśla jak funkcja plot(X,Y) ale dodatkowo pozwala wybierać kolor, rodzaj linii i symbole punktów – patrz tabela poniżej.

Kolor
y – yellow
m – magenta
c – cyan
r – red
g – green
b – blue
w – white
k – black

Symbole punktów
. – point
o – circle
x – x-mark
+ – plus
* – star
s – kwadraty
d – romb
v – trójkąt w dół
^ – trójkąt w górę
< – trójkąt w lewo
> – trójkąt w prawo
p – pięciokąt
h – sześciokąt

Rodzaj linii
- – ciągła
: – kropkowana
-. – kropka-kreska
-- – kreskowana

Przykład:

```
>> plot(1:10,y) % wykreśla wektor od 1 do 10 w funkcji wektora y
>> plot(1:10,y, 'bx') – j.w. ale dodatkowo wykreśla go w kolorze niebieskim zaznaczając
      punkty krzyżykami.
>> plot(1:10,x, 'bx ', 1:10,y, 'r*') – wykreśla dwa wykresy na jednym
```

Temat 37
Wykresy dwuwymiarowe funkcji – funkcja *fplot*

fplot(F,P) – funkcja wykreśla funkcję F daną w postaci łańcucha w przedziale P.
Listę funkcji matematycznych predefiniowanych w MATLABie można uzyskać poprzez polecenie >> help elfun (funkcje podstawowe) i >> help specfun (funkcje specjalne)

Przykład:

```
>> fplot('2*sin(x) ',[0 2*pi]) % funkcja 2*sin(x) w przedziale od 0 do 2π
```

Temat 38
Wykresy dwuwymiarowe funkcji – funkcja *ezplot (*funkcja obecnie nie jest rekomendowana przez mathworks*)*

Bardziej ogólnymi funkcjami służącymi do rysowania wykresów także dla dwu zmiennych są funkcje typu ez*. Jedną z nich jest funkcja ezplot.
ezplot(F,P) – funkcja wykreśla funkcję F w przedziale P.
Inne funkcje należące do kategorii ez* to: ezcontour, ezmesh, ezmeshc, ezpolar.

Przykład:

```
>> ezplot(x, 2*y, [0,2*pi]) % wykres funkcji parametrycznej typu x=x(t), y=y(2t), t∈[0,2π]
```

Przykład:

```
>> xt=@(t) cos(3*t)
>> yt=@(t) sin(2*t)
>> fplot(xt,yt);   % wykres funkcji parametrycznej
```

Temat 39
Wykresy dwuwymiarowe funkcji – funkcje *hist, starsi, bar, steam*

hist(x, m)	Wykreśla histogram z podziałem na m przedziałów.
stairs()	wykreśla wektor w postaci schodków od największego do najmniejszego elementu
bar(x)	wykreśla wektor w postaci słupków (bar)
stem(x)	wykreśla wektor w postaci linii pionowych (ystem)
errorbar	Wykreśla wektora wartości z błędami

Uwaga: Wywołanie n=hist(X) nie wyświetla wykresu, ale zlicza ilość elementów wektora w 10 równych przedziałach. Przedziały są tworzone na podstawie najmniejszej i największej wartości wektora

Temat 40
Rysowanie wielu wykresów na wspólnym wykresie graficznym – funkcja *hold*

```
>> hold on % wstrzymuje czyszczenie okna graficznego
>> hold off % przywraca tryb domyślny (każdorazowe czyszczenie okna)
>> ishold % testuje tryb rysowania wykresów
```

Temat 41
Otwieranie wielu okien graficznych – funkcje *figure, close, clg, cla*

```
>> figure % otwiera nowe okno graficzne
>> figure(n) % uaktywnia okno graficzne o danym parametrze,
>> close % zamyka okno aktywne lub okno z zadanym parametrem.
>> cla % czyści bieżący wykres
>> clf % czyści aktywne okno graficzne
```

Temat 42
Wykreślanie niezależnych wykresów w jednym oknie graficznym – funkcja *subplot*

Funkcja subplot służy do podziału okna graficznego na mniejsze fragmenty. Podziału można dokonać albo w układzie macierzowym albo podając dokładne wymiary wykresu.

Przykład:

```
>> subplot(m,n,p) % dzieli okno graficzne na M kolumn i N wierszy (M,N<9). P oznacza
      numer aktualnego wykresu. Można też wywołać jako subplot(mnp)

>> subplot('position',[lewy dolny szerokość wysokość]) % w aktywnym oknie graficznym
      tworzy nowy wykres w zadanym podoknie. Lewy, dolny – współrzędne lewego dolnego
      rogu podokna. Szerokość, wysokość – rozmiary podokna. Wszystkie rozmiary podaje
      się w stosunku do całości okna unormowanego do 1, np.: [0.5 0.5 0.5 0.5]
```

Temat 43

Skalowanie wykresów – funkcje *axis* i *log*-i

axis('auto')	domyślny tryb skalowania
axis([xmin, xmax, ymin, ymax])	wykreśla wykres w zadanych przedziałach osi X i Y
axis('off')	ukrywa osie
axis('on')	przywraca wyświetlanie osi
axis('equal')	osie mają proporcjonalne jednostki na obu osiach X i Y

loglog(x)	skala logarytmiczna na obu osiach
semilogx(x)	skala logarytmiczna na osi X
semilogy(x)	skala logarytmiczna na osi Y

Temat 44

Opisywanie wykresów

```
>> plot(x,y, 'r ') % wykres funkcji
>> title('To jest wykres') % Tytuł wykresu
>> grid off % wyłączenie wyświetlania siatki
>> xlabel('oś X') % podpis osi X
>> ylabel('oś Y') % podpis osi Y
>> text(2,4, 'tu jest punkt') % tekst wstawiony w punkcie (2,4)
```

Temat 45

Niestandardowe znaki w opisie wykresów

Do wypisywania niestandardowych znaków wykorzystywana jest składnia TeX.

```
>> text(1,1, '\alpha^{3/2}') % wypisanie w punkcie (1,1) tekstu α^3/2
```

Temat 46

Zmiana pozostałych parametrów funkcji graficznych.

```
>> plot(x,y, 'LineWidth',4, 'MarkerSize',10)
```

Informacje o poszczególnych elementach wykresu można znaleźć w helpie, np: *help line*

Temat 47

Modyfikacja wykresów w oknie graficznym

Temat 48

Wybór punktów na wykresie – Funkcja *ginput*

Gdy zachodzi potrzeba wybrania jakiś punktów na wykresie w celu dalszego przetwarzania korzystamy z funkcji *ginput*.

Przykład:

```
>> [x,y]=ginput(2); % pobranie współrzędnych 2 punktów wskazanych w oknie graficznym
```

WIZUALIZACJA DANYCH
WYKRESY TRÓJWYMIAROWE

Temat 49
Funkcja meshgrid

Funkcja meshgrid – tworzy macierze opisujące położenie węzłów siatki prostokątnej. Służy do przygotowania danych niezbędnych do stworzenia większości wykresów 3D.

Przykład:

```
>> [X,Y]=meshgrid(x,y); % tworzy macierze X, Y na podstawie wektorów z węzłami siatki x, y
>> [X,Y]=meshgrid(x); % j.w. ale y=x
>> [X,Y,Z]=meshgrid(x,y,z) % tworzy 3 macierze wykorzystywane do wykresów
      volumetrycznych
```

Temat 50
Funkcja *mesh*

Mesh(X,Y,Z) – funkcja mesh rysuje siatkę opisaną przez macierze X,Y,Z. Gdzie macierze X, Y podają współrzędne punktów siatki a dane w macierzy Z określają wartość funkcji w punkcie (x,y).
Mesh(X,Y,Z,c) – c – indeksy kolorów w aktualnej mapie kolorów.

Przykład:

```
>> [x,y] = meshgrid(-3:.125:3); % generacja siatki
>> z = peaks(x,y); % tworzenie wartości funkcji w punktach (x,y)
>> mesh(x,y,z) % tworzy wykres 3D
```

Temat 51
Inne wykresy 3D typu oparte na funkcji meshgrid

contour3	Wykres konturowy
ezmesh	Wykres siatkowy
ezsurf	Wykres – powierzchnia
mesh	Wykres siatkowy
meshc	Wykres jak mesh + poziomice
meshz	Wykres jak mesh + zasłony na końcach
ribbon	Wykres wstążkowy
Surf	Wykres powierzchniowy
Surfc	Wykres powierzchniowy + poziomice
Surfl	Wykres powierzchniowy + cieniowanie
Waterfall	Wykres plasterkowy

Temat 52
Inne wykresy 3D

bar3	Wykres słupkowy
ezplot3	Wykres parametryczny
isosurface	Izowarstwy dla danych 3D
plot3	Linia w 3 wymiarach
scatter3	Wykres typu scatter
slice	Przekrój przez wykres wolumetryczny

Przykład:

```
>> t = 0:pi/50:10*pi;
>> plot3(sin(t),cos(t),t) % linia śrubowa
>> ezplot3('sin(t)','cos(t)','t',[0,6*pi]) % linia śrubowa
>> a=rand(5); generowanie danych
>> bar3(a) % wykres słupkowy
```

Temat 53
Obiekty 3D

cylinder	Generacja walca
Elipsoid	Generacja elipsoida
fill3	Generacja wielokąta
sphere	Generacja kuli

Przykład:

```
>> sphere % wyświetla sferę
>> [x,y,x]=sphere; % zwraca 3 macierze z danymi do wyrysowania kuli za pomocą funkcji
      mesh lub surf
```

Temat 54
Widoki wykresów 3D

zlabel	Opis osi z
view	Zmiana domyślnego punktu obserwacji
view(azymut, elewacja)	Okresla punkt obserwacyjny za pomocą azymutu i elewacji
view(x,y,z)	Okresla punkt obserwacji w układzie kartezjańskim
view(2)	Obserwacja azymut=0, elewacja=90
view(3)	Domyślny punkt obserwacji: azymut=-37.5 , elewacja= 30

hidden on	Wyświetlanie ukrytych krawędzi
hidden off	Domyślny, ukrywa niewidoczne krawędzie

shading flat	Powierzchnia z dyskretnymi kolorami
shading intern	Powierzchnia z wypełnieniem kolorami interpolowanymi
shading faced	Powierzchnia z dyskretnymi kolorami i siatką

caxis	Przeskalowanie kolorów

Temat 55
Wizualizacja 3D

camlight	Definiuje oświetlenie we współrzędnych kamery
Light	Definiuje obiekt świecący
lightangle	Położenie kamery we współrzędnych sferycznych
lighting	Algorytm liczenia oświetlenia: flat, gouraud, phong, none
material	Określa właściwości odbiciowe materiału: shiny, dull, metal, default

Przykład:
sphere
lightangle(-5,130)
lighting gouraud

Temat 56
Wizualizacja wolumetryczna

Dane trójwymiarowe możemy przedstawić albo przez wyświetlanie poszczególnych przekrojów, powierzchni o stałej wartości lub przepływów.

coneplot	Pole wektorowe 3D
contourslice	Kontury w warstwach
isosurface	Powierzchnia o stałej wartości (izopowierzchnia)
slice	Płaszczyzna przekroju
streamline	Linie przepływu
streamparticles	Cząstki wraz z liniami przepływu
streamribbon	Wstęgi zgodne z przepływem
streamslice	Przepływ w warstwach lub na powierzchniach
streamtube	Przepływ pokazany za pomocą walcy
quiver	Pole wektorowe 2D

Wizualizacja pola wektorowego w punktach (x,y) o wartościach (u,v): **quiver(x,y,u,v)**

Przykład:
```
[x,y] = meshgrid(0:0.2:2,0:0.2:2);
u = cos(x).*y;
v = sin(x).*y;
quiver(x,y,u,v)
```

Linie prądu: streamline(X,Y,Z,U,V,W,startx,starty,startz)
Gdzie startx,starty,starty definiuje początk linii prądu

Przykład:
```
[x,y] = meshgrid(0:0.1:1,0:0.1:1);
u = x;
v = -y;
quiver(x,y,u,v)
startx = 0.1:0.1:1;
starty = ones(size(startx));
streamline(x,y,u,v,startx,starty)
```

Przykład streamtube:
Szerokość tuby jest proporcjonalna do znormalizowanej dywergencji pola wektorowego
```
load wind
[sx sy sz] = meshgrid(80,20:10:50,0:5:15);
verts = stream3(x,y,z,u,v,w,sx,sy,sz);
div = divergence(x,y,z,u,v,w);
streamtube(verts,x,y,z,-div);
% Define viewing and lighting
view(3)
shading interp
```

Przykład: slice(X,Y,Z,V,sx,sy,sz)
```
[x,y,z] = meshgrid(-2:.2:2,-2:.25:2,-2:.16:2);
v = x.*exp(-x.^2-y.^2-z.^2);
xslice = [-1.2,.8,2];
yslice = 2;
zslice = [-2,0];
slice(x,y,z,v,xslice,yslice,zslice)
```

PODSTAWY PROGRAMOWANIA: SKRYPTY I FUNKCJE

Temat 57
Tworzenie skryptów

Skrypt jest plikiem tekstowym zawierającym zestaw funkcji i poleceń Matlaba. Pliki skryptowe mają rozszerzenie **.m**.
Pliki skryptowe można tworzyć w każdym edytorze tekstowym. Najwygodniej wykorzystać edytor Matlaba. Dostęp do edytora jest możliwy przez File -> New-> M-file lub przez odpowiednią ikonę.

Temat 58
Opisywanie skryptów

Każdy skrypt powinien mieć krótki opis zawartości i działania. Opis umieszcza się za znakiem %. Ze względów praktycznych opis należy umieszczać za podwójnym znakiem procenta (%%).
Począwszy od Matlaba 7 znak %% oznacza nową fragment kodu.
Znaki %% oraz % są też inaczej traktowane w czasie konwersji skryptu do html-a.
Opis pliku można wywołać w Matlabie przy pomocy polecenia *help nazwa_skyptu.* Za opis pliku traktowane są pierwsze linie komentarza nieprzerwane liniami innego typu.

Przykład:

```
%% To jest test opisu skryptu piewszy_skrypt.m
%% Jestem w skrypcie
% czy widać tę linię?
a=5; % jakaś komenda
% czy widać tę linię
```

Temat 59
Zmienne w skryptach Matlaba

Skrypty do przechowywania zmiennych używają przestrzeni roboczej Matlaba. Z jednej strony nie trzeba definiować mu zmiennych, ale istnieje niebezpieczeństwo użycia i zamazania zmiennych istniejących już w przestrzeni roboczej.

Temat 60
Wypisywanie kroków wykonywanych w skrypcie na ekran.

Analogicznie do poleceń wypisywanych w Oknie Poleceń, polecenia wykonywane w skrypcie dają echo na ekranie. Aby przyśpieszyć pracę skryptów oraz dla zapewnienia uniwersalności (dobry nawyk dla programistów) należy wszystkie polecenia wykonywać z opcją ukrywania echa (o ile celem pliku nie jest narysowanie wykresu). Do ukrywania echa stosuje się średnik na końcu linii polecenia – patrz Temat 7.

Temat 61
Tworzenie funkcji

Funkcja tak jak skrypt jest plikiem tekstowym zawierającym zestaw funkcji i poleceń Matlaba i zaczynać się powinna od słowa kluczowego **function**. Pliki funkcji mają również rozszerzenie **.m**.

UWAGA: Ważne jest aby nazwa funkcji i nazwa pliku były takie same.

Pliki funkcji można tworzyć w każdym edytorze tekstowym. Najwygodniej wykorzystać edytor Matlaba.
Podstawową różnicą miedzy funkcją a skryptem jest sposób przechowywania danych. Skrypt czyni to w przestrzeni roboczej, natomiast funkcja przechowuje je poza przestrzenia roboczą, co pozwala na dublowanie nazw zmiennych z przestrzenią roboczą. Inaczej mówiąc funkcja jest hermetyczna i pokazuje na zewnątrz tylko dane wyjściowe, lub zmienne specjalnie udostępnione przy pomocy operatora global.

Temat 62
Szkielet funkcji

```
function [x,y,z]=nazwa_funkcji(a,b,c,d)
%% [x,y,z]=nazwa_funkcji(a,b,c,d)
%% Funkcja zwraca 3 wektory x,y,z dla danych parametrów wejściowych a,b,c,d

%%Koniec nazwa_funkcji.m
```

Funkcja powinna posiadać następujące elementy:

- nagłówek funkcji – definicje parametrów funkcji - argumentów, (a, b, c, d – w naszym szkielecie) oraz parametrów wyjścia - wartości, (x, y, z – w naszym szkielecie);
- komentarz z opisem do help-u – opisuje co funkcja robi, opisuje argumenty funkcji oraz wartości wyjściowe;
- analiza liczby parametrów wejściowych – moduł funkcji analizuje liczbę parametrów wejściowych, czy jest ich wystarczająco dużo do wykonania funkcji i czy ewentualnie można przyjąć wartości domyślne dla niepodanych parametrów (na razie się tym nie zajmujemy);
- analiza własności parametrów wejściowych – moduł funkcji sprawdza czy wartości wprowadzonych argumentów umożliwiają poprawne wykonanie funkcji (na razie się tym nie zajmujemy);
- implementacja algorytmu – zapewnia poprawność obliczeń numerycznych i przygotowuje wartości wyjściowe.

Funkcje typu Anonymous @

```
>>SQR=@(x)  x.^2;
>>S=@(x,y) (x.^2+y.^2);
>>F=@(x,y) [x.^2+y,y-x];
```

Funkcja powinna posiadać następujące elementy:

Uwaga: Począwszy od wersji Matlak 2016b możliwe jest definiowanie funkcji wewnątrz skryptu.

Temat 63
Postępowanie przy pisaniu funkcji

Najwygodniej najpierw napisać skrypt a po przetestowaniu przerobić go w funkcję.

Temat 64
Przykład funkcji i wywołania funkcji

Zawartość pliku przykład_1.m:

```
function [x,y]=przyklad_1(a,b)
%% [x,y]=przyklad_1(a,b)
%% Funkcja rysuje wykres funkcji y=a*cos(x+(pi/b))
%% zwraca 2 wektory x – wektor zmiennej x, y - wektor z wynikami funkcji
%% dla danych parametrów a, b. Funkcja rysuje wykres funkcji w aktywnym oknie.
x = 0:0.001:2*pi;
y = a.*cos(x+(pi./b));
plot(x,y);
%Koniec przyklad_1.m
```

Wywołanie funkcji z Okna Poleceń:

```
>> przyklad_1(2,2); % rysuje wykres funkcji
>> [ax,ay]=przyklad_1(2,2);  % oprócz wykresu wyprowadza do przestrzeni roboczej dwa
      wektory ax, ay.
>> parametr1=3;, parametr2=4;
>> [ax,ay]=przyklad_1(parametr1, parametr2); % j.w.
```

Temat 65
Pod funkcje

W jednym pliku zawierającym funkcję można umieścić więcej funkcji. Przy czym tylko pierwsza funkcja jest widoczna na zewnątrz. Wszystkie pozostałe funkcje mogą być wywoływane tylko w obrębie danego pliku.

DEBUGOWANIE W MATLABIE

Debugowanie w matlabie może być realizowane za pomocą edytora lub komend. Wyróżnia się trzy typy przerwań wykonania programu: **standardowe, warunkowe oraz związane z błędem**. W edytorze skryptu lub funkcji po lewej stronie gdzie znajdują się numery linie a następnie znak „—„ prawym przyciskiem ustawiamy odpowiedni kolor przerwania:
- czerwony odpowiada standardowemu przerwaniu
- żółty warunkowemu (przy czym ustawiamy odpowiedni warunek Set/Modify Condition).
Ustawienie przerwania związanego z błędem odbywa się poprzez przycisk RUN i wybranie odpowiedniej opcji:
- pause on Errors to pause on all errors,
- pause on Warnings to pause on all warnings
- pause on NaN or Inf to pause on NaN (not-a-number) or Inf (infinite) values.
W przypadku linii komend mamy następujące możliwości:

- **dbstop** in myprogram at 2
oznacza przerwanie standardowe w linii nr 2 w programie myprogram.m

- **dbstop** in myprogram at 6 if n>=4
oznacza warunkowe przerwanie programy w linii nr 6 jeśli parametr n>=4 (np. w pętli)
- **dbstop if error**
oznacza przerwanie, jeśli wystąpi błąd
Po ustawieniu funkcji **dbstop** uruchamiamy program. Po przerwaniu wykonywania programu w oknie poleceń pojawia się znak K>> oznaczający tryb debugowania i możliwości wykonywania komend z okna poleceń. Wyłączenie wszystkich przerwań odbywa się za pomocą komendy:
dbclear all in myprogram

Dodatkowa możliwości to zatrzymanie programu przy użyciu „**keyboard**" z możliwością wykonania komendy z linii poleceń. Zamkniecie modu debugowego odbywa się w tym przypadku za pomocą komendy **dbquit**.

INSTRUKCJE W MATLABIE

Temat 66
Instrukcja *for*

Instrukcja for pozwala na powtarzanie wybranego fragmentu kodu określoną ilość razy.
Szablon instrukcji for (uwaga na przecinek):

```
....
for zmienna_iterowana = macierz_wartości ,
.....
Kod do wielokrotnego powtarzania
....
end
.....
```

Pętle w wybranych przypadkach można przerywać przy pomocy instrukcji **break**.

Przykład:

```
a=zeros(10,5); % alokacja pamięci
for i=1:10,
   for j=1:5,
      a(i,j)=i*j;
   end
end
```

Instrukcja for dla obliczeń równoległych przyjmuje postać **parfor**.
Szablon instrukcji for (uwaga na przecinek):

```
....
parfor zmienna_iterowana = macierz_wartości,
.....
```

```
Kod do wielokrotnego powtarzania
....
end
.....
```

Przykład:

```
parpool(3)
parfor i=1:3
c(:,i)=eig(rand(1000));
end
```

Temat 67
Instrukcja *while*

While stanowi pętle warunkową, fragment kodu w pętli będzie wykonywany dopóki jest spełnione wyrażenie warunkowe.
Szablon instrukcji while (uwaga na przecinek):

```
....
while wyrażenie_warunkowe,
.....
Kod do wielokrotnego powtarzania
....
end
.....
```

Przykład:

```
licznik1=0; , licznik2=0; , suma=0; % definicja stałych
while (licznik1<10 & licznik2<10), % znak & oznacza and, opis – help ops
   licznik1=licznik1+0.1;
   licznik2=licznik2+0.2;
   suma=licznik1+licznik2;
end
```

Temat 68
Instrukcja warunkowa *if*

Instrukcja pozwala na wykonanie jednego z kilku fragmentów kodów zawartego pomiędzy instrukcjami **if**, **elseif**, **else**. Wybór realizowanego kodu zależy od spełnienia odpowiednich wyrażeń warunkowych, gdy żadne z nich nie jest spełnione jest wykonywany kod występujący za operatorem **else**.
Szablon instrukcji if:

```
If wyrażenie_warunkowe_1
   Kod wersja 1
elseif wyrażenie_warunkowe_2
   Kod wersja 2
elseif wyrażenie_warunkowe_3
   Kod wersja 3
```

```
.....
else
   Kod wersja N
end
```

Przykład:

```
%% y=a*x^2+b*x+c
a=1; , b=2; , c=3; % definicja stałych
wyznacznik=b^2-4*a*c; % np. wyznacznik równania kwadratowego
if wyznacznik>0
   x1=(-b+sqrt(wyznacznik))/(2*a); , x2=(-b-sqrt(wyznacznik))/(2*a);
elseif wyznacznik==0
   x1=-b/(2*a); , x2=x1;
else
   x1=NaN; , x2=NaN;
end
```

Temat 69
Instrukcje *break* i *return*

Obie instrukcje powodują przerwania wykonywania kodu. Funkcja **break** powoduje wyskoczenie z najgłębiej zagnieżdżonej pętli do wyższej pętli. Funkcja **return** powoduje natychmiastowe opuszczenie danej funkcji lub skryptu i powrót do miejsca jej wywołania.

Temat 70
Instrukcja *switch-case*

W przypadku listy znanych argumentów wywołania wygodnie jest skorzystać z funkcji switch-case.
Szablon instrukcji switch-case:

```
switch p
   case 1
      instrukcja 1
   case 2
      instrukcja 2
   otherwise
      inna instrukcja
end
```

Instrukcja try

```
try
 instrukcja 1    (jesli zakonczy się błędem wykonana będzie isntrukacja 2)
catch
instrukcja 2
end
```

INNE PRZYDATNE FUNKCJE

Temat 71
Funkcje pomiaru czasu

cputime	Czas CPU który upłynął od uruchomienia Matlaba (ogólnie do pomiaru czasu)
tic	Start stopera
toc	Zatrzymanie stopera
etime	Czas, który upłynął pomiędzy dwoma podanymi datami w formie wektorów
pause	Zatrzymanie na x sekund – zwykle oczekiwanie na odpowiedź użytkownika przy programach interakcyjnych

Temat 72
Testowanie funkcji – czas wykonywania funkcji – *tic* i *toc*

W przypadku testowania programów najwygodniej używać funkcji tic i toc.

Przykład:

```
tic
testowana_Funkcja
toc
```

```
>> Elapsed time is 2.188000 seconds.
```

Temat 73
Funkcje daty i czasu

Funkcje czasu i daty znajdują się w grupie funkcji timefun – help timefun

now	Aktualna data jako liczba dni od 01.01.0
date	Aktualna data i godzina jako zmienna łańcuchowa
clock	Aktualna data i godzina jako wektor

datenum	data jako liczba dni od 01.01.0
datestr	data jako zmienna łańcuchowa
datevec	Transformacja składników daty do postaci wektora

calendar	Kalendarz
weekday	oblicza dzień tygodnia dla podanej daty
eomday	zwraca liczbę dni w miesiącu w podanym roku i miesiącu
datetick	formatowanie daty

Temat 74
Funkcje w Matlabie – ciąg dalszy – zmienne globalne *global*

Przypomnienie: zmienne w funkcji są lokalne – nie widać ich na zewnątrz. Tak samo zmienne w obszarze roboczym są niewidoczne dla funkcji chyba, że są jej parametrem wejściowym. Nawet wtedy jednak są przekazywane przez wartość, także ich wartość modyfikowana wewnątrz funkcji wróci do wartości początkowej po wyjściu z funkcji. Jednak czasami takie

ograniczenia nie są wygodne. Gdy chcemy, aby zmienne z przestrzeni roboczej były dostępne wewnątrz funkcji bez definiowania ich jako parametry funkcji, wtedy deklarujemy je jawnie w przestrzeni roboczej oraz w samej funkcji poprzez ***global***. Takie działanie jest jednak niebezpieczne, bo może dojść do konfliktu nazw pomiędzy funkcją i przestrzenią roboczą, lub niepożądaną zmiana ich wartości.

Przykład:

```
function [.....]=fun(....)
%% opis funkcji
global a1 a2 a3;
.....
% koniec funkcji
```

```
%% w przestrzeni roboczej
global a1 a2 a3;
a1=....
a2=.....
a3=.....
```

Temat 75

Funkcje w Matlabie – ciąg dalszy – funkcja *feval*

Często istnieje potrzeba, aby dana funkcja matlabowska (plik *.m) była w stanie przeprowadzić obliczenia dla dowolnych funkcji matematycznych zdefiniowanych poza plikiem *.m. Wtedy stosuje się funkcję *feval*.
Definicja funkcji feval:

```
>> y = feval(Nazwa_funkcji, x1 .....xn) % Nazwa_funkcji  - zmienna łańcuchowa
%% , x1 .....xn – zadane argumenty funkcji
```

Przykład:

```
y= feval('cos',[0:0.01:pi]);
```

Przykład: Funkcja suma_ciagu, która wylicza sumę n wyrazów dowolnego ciągu

```
function s=suma_ciagu(n,ciag)
%% s=suma_ciągu(liczba wyrazów, 'nazwa_funkcji')
i=[1:n];
s=sum(feval('ciag',i)
% koniec funkcji suma_ciagu
```

```
function [a]=ciag(n)
%% [a]=ciag(n) – tu definiuję jak wygląda n-ty wyraz ciągu
a=0.5 .^n
%koniec funkcji ciag
```

Temat 76

Operacje łańcuchowe

```
>> a='to jest lancuch';  % definicja łańcucha i przypisanie go zmiennej a
>> b='drugi';
>> c=strat(a, b);  % połączenie łańcuchów. Zmienna c = `to jest lancuch drugi`
>> d=[a b]; % j.w.
```

```
>> t=num2str(15.4); % Zamiana liczby na łańcuch
>> d=str2num(`15.4`); % Zmiana łańcucha na liczbę
```

Temat 77
Rekurencja

Rekurencja jest eleganckim, ale bardzo kosztownym sposobem programowania. Można ją stosować tam gdzie mamy do czynienia z zależnościami typu f(n+1)=g(f(n)).

Przykład:

```
function s=silnia(n)
%% Obliczanie silni metodą rekurencyjną
s=1;
for k=2:n,
   s=k*silnia(k-1); % funkcja wywołuje sama siebie
end
```

OKNA DIALOGOWE

Temat 78
Wybrane funkcje do wyświetlania komunikatów na ekranie komputera

inputdlg – otwiera okno służące do wprowadzania danych

msgbox – wypisuje komunikat w formie błędu, ostrzeżenia lub pomocy

errordlg – wypisuje komunikat o błędzie

listdlg – umożliwia wybór z listy

questdlg – definiuje dowolne okno dialogowe

uigetfile – otwiera okno systemem plików i zwraca nazwę pliku oraz katalog

uiputfile – otwiera okno systemem plików i zwraca nazwę pliku oraz katalog

ROZWIĄZYWANIE UKŁADÓW RÓWNAŃ LINIOWYCH

Temat 79
Rozwiązywanie układów równań liniowych

Matlab ma bardzo rozwinięte algorytmy rozwiązywania równań liniowych. W zależności od potrzeb można używać metod zaawansowanych (Matlab stara się dobrać metodę w tle) lub ręcznie poprzez Metodę Gaussa, uzyskiwanie rozkładu macierzy na macierze trójkątne itd.)

Układ równań linowych można zapisać wektorowo w postaci:
A*x=b, gdzie A macierz współczynników, x – wektor zmiennych [x1...xn], b – wektor wartości równań [b1...bm].
Uwaga: z rozwiązaniem układu równań nie ma problemu pod warunkiem, że układ nie jest sprzeczny, jest dobrze określony, i jest liniowo niezależny. W przeciwnym wypadku trzeba stosować bardziej zaawansowane metody obliczeń.
Do sprawdzania uwarunkowania macierzy służy funkcja ***cond(a)***. Duże wartości funkcji ***cond*** świadczą o złym uwarunkowaniu – to wpływa na dokładność obliczeń numerycznych.

Temat 80
Metody obliczania układów równań

- operator dzielenia lewostronnego: x=A\b – praktycznie jest tu stosowana metoda eliminacji Gaussa z częściowym wyborem elementu głównego
- przez mnożenie wektora wynikowego przez macierz odwrotna współczynników x=inv(A)*b

Przykład:

```
% rozwiązanie układu równań w postaci [a]*[x]=[b]
>> a = [ 1 -4  3; 3  1 -2; 2  1  1]; % definicja macierzy współczynników
>> b = [ -7; 14; 5]; % definicja wektora wyników
>> x = inv(a)*b; % rozwiązanie metodą odwrócenia macierzy
>> x = a\b; rozwiązanie metodą dzielenia lewotronnego
```

Uwaga: równanie x=b/A daje wynik rozwiązania układu równań w postaci x*A=b.

Temat 81
Eliminacja Gaussa

Podstawowa metodą rozwiązywania układów liniowych jest metoda eliminacji Gaussa – tzw. rozkład LU. Polega on na znalezieniu macierzy L i U takich, że A=L*U, gdzie U jest macierzą trójkątną górną, a L macierzą trójkątną dolną.
W Matlabie eliminację Gausa przeprowadza funkcja ***lu***.
Przy wywołaniu [L,U]=lu(A) U jest macierzą trójkątną górną, ale L nie zawsze będzie macierzą trójkątną dolną.
Przy wywołaniu [L,U,P]=lu(A) U jest macierzą trójkątną górną, L nie zawsze będzie macierzą trójkątną dolną, a P macierzą permutacji (zmienia kolejność wierszy w macierzy A). Zachodzi tu zależność L*U=P*A.

Temat 82

Inne funkcje związane z układami równań liniowych

det	wyznacznik macierzy
inv	odwrotność macierzy
eig	wartości własne
chol	rozkład Cholesky'ego, rozkład macierzy A na macierz L i L' takie, że A=L'*L

ROZWIĄZYWANIE RÓWNAŃ W SPOSÓB SYMBOLICZNY

Deklaracje zmiennych symbolicznych:
syms a b c x
zapisywanie równań:
eq=a*x^2+b*x+c==0
podstawienie do równania: **subs**(eq,val)
rozwiązywanie równań:
S=**solve**(eq)
S=solve(eq,x) – rozwiązanie ze względu na zmienną x
S=solve(eq,x,'Real','true) – tylko rzeczywiste rozwiązania
solve(x>0,x<2)
assume(x/2,'integer')
lhs(eqn) – lewa strona równania
rhs(eqn) – prawa strona równania

Równania różniczkowe rozwiązujemy funkcją **dsolve**.
Np. równanie w postaci dy/dt=ay
syms y(t) a
eq=diff(y,t)==ay
S=dsolve(eq)

Równanie II stopnia: d2y/dt2=ay
Eq=diff(y,t,2)==a*y;
dsolve(eq)
Ewentualnie z warunkami początkowymi
cond=y(0)==5
dsolve(eq, cond)

Całkowanie symboliczne przy użyciu funkcji **int**

Przykład:
syms x
expr=-2x/(1+x.^2)^2
całka nieoznaczona
F=int(ext)
Całka oznaczona
F=int(ext,[0 1])
Gdy brak rozwiązania można wykonać obliczenia numeryczne vpa(F)

ROZWIĄZYWANIE UKŁADÓW RÓWNAŃ NIELINIOWYCH

Temat 83
Rozwiązywanie równań nieliniowych

Przy rozwiązywaniu równań poszukujemy pierwiastków równań, maksimów i minimów funkcji. Pierwiastki rzeczywiste równania, (czyli miejsca zerowe) –> f(x)=0.
W Matlabie funkcja ***fzero*** wyszukuje pierwiastek równania w pobliżu zadanej wartości zmiennej. Czyli do znalezienia wszystkich pierwiastków równania trzeba podać okolice gdzie ma on występować.
Minimum lokalne funkcji poszukuje się analogicznie do pierwiastków przy pomocy funkcji ***fminbnd***.
Maximum lokalne funkcji poszukuje się przez poszukiwanie minimum funkcji odwrotnej do danej, czyli -> -f(x).
W przypadku wielomianów wartości funkcji wyszukuje się poprzez funkcje ***roots(c),*** gdzie c jest wektorem współczynników wielomianu.
W celu obliczenia wartości wielomianu korzystamy z funkcji ***polyval(c,x)***, gdzie *x* jest liczbą, wektorem lub macierzą dla której liczymy wartości wielomianu.

Przykład:

```
>> x=fzero('sin',10); % szuka miejsca zerowego funkcji sinus w okolicach 10
>> m=fminbnd('sin',10,11); % szuka najmniejszej wartości funkcji sinus w przedziale (10,11)
>> p=[3,-2,4,1]; % definicja wielomianu p=3x^3-2x^2+4x+1
>> r=roots(p); % zwraca pierwiastki równania
>> w=polyval(p,2); % zwraca wartość wielomianu 3*2^3-2*2^2+4*x+1
```

Funkcja **fsolve** rozwiązuje nieliniowe układy równań wielu zmiennych $F(x) = 0$
Podstawowe wywołanie
x = fsolve(fun,x0)
gdzie fun jest nazwa funkcji lub uchwytem do funkcji @, zaś x0 jest wektorem startowym opcjonalny trzeci parametr funkcji **fsolve** pozwala ustawiać dodatkowe parametry metody minimalizacyjnej oraz wypisywanie kolejnych integracji na ekran np.
option=optimoptions('fsolve','Display','iter');

Przykład:

$2x_1-x_2=\exp(-x_1)$
$-x_1+2x_2=\exp(-x_2)$
Przepisując układ do postaci $F(x) = 0$:
$2x_1-x_2-\exp(-x_1)=0$
$-x_1+2x_2-\exp(-x_2)=0$
Definiujemy funkcję

```
>>function F = myfun(x)
>>F = [2*x(1) - x(2) - exp(-x(1));
 >>     -x(1) + 2*x(2) - exp(-x(2))];
```

Definiujemy dodatkowe opcje w celu wyświetlenia kolejnych kroków iteracji

```
>>options = optimoptions('fsolve','Display','iter');
```

Jako wektora startowy przyjmujemy

```
>> x0 = [-5 -5].
>>[x,fval] = fsolve(@myfun,x0,options)
```

INTERPOLACJA I APROKSYMACJA FUNKCJI

Temat 84
Interpolacja

Interpolacją nazywamy zadanie znalezienia krzywej przechodzącej przez zadane punkty. Te zadane punkty nazywa się węzłami interpolacji.
W Matlabie stosuje się kilka metod interpolacji: wielomianami pierwszego i trzeciego stopnia, metodą najbliższych sąsiadów oraz za pomocą funkcji sklejanych. Interpolacje stosuje się do tzw. zagęszczania tabel. Np. mamy tabelę z krokiem dla osi x równym 1, a chcemy stworzyć tabelę z krokiem 0.2.

Temat 85
Funkcja *Interp1*

yi=interp1(x, y, xi, 'metoda') gdzie:
x, y – wektory współrzędnych węzłów interpolacji,
xi – wektor punktów na osi X dla których będą obliczane interpolowane wartości yi
metoda:

'linear'	funkcja łamana
'spline'	funkcja sklejana 3-go stopnia
'cubic', 'pchip'	wielomian 3-go stopnia
'nearest'	funkcja najbliższego sąsiedztwa

Przykład:

```
% Interpolacja funkcji sinus, na wykresie węzły zaznaczone są punktami, dodatkowo
% rysowana jest wzorcowa funkcja.
>> x=0:10; y = sin(x); xi = 0:.25:10;
>> yi = interp1(x, y, xi);
>> plot(x, y, 'o', xi, yi, sin(xi))
```

Temat 86
Funkcje interp2, interp3, interpn

Funkcje interpolujące w 2 i 3 wymiarach dla danych określonych na regularnej siatce. Do generacji siatki należy używać funkcji **meshgrid**
Funkcja interpn wymaga generacji siatki przy użyciu funkcji **ndgrid**

Przykład:

```
>> [x,y]=meshgrid(xv,yv);  xv i yv są wektorami
>>zi=interp2(x,y,z,xi,yi); % x, y, z – dane funkcji, xi, yi – nowe zagęszczone punkty
>> [x,y,z]=meshgrid(xv,yv,zv);
>> vi=interp3(x,y,z,v,xi,yi,zi); % x, y, z, v – dane funkcji, xi, yi, zi – nowe zagęszczone punkty
```

Funkcje griddata
Funkcja używana jest do interpolacji wartości zdefiniowanych na nieregularnej siatce na siatkę regularną

Przykład:

```
>> V = griddata(x,y,v,X,Y,metoda)
% x, y wektory definiujące położenie punków dla których zdefiniowane są wartość v
%  X , Y wektory lub macierze siatki do których interpoluje się dane zdefiniowane dla wektora v
Metoda:
 'nearest','linear','natural','cubic'
```

Funkcja scatterInterpolant - interpolacja 2-D lub 3-D danych rozproszonych (analogiczna griddedInterpolant)
Przykład:

```
x = rand(10,1);
y = rand(10,1);
z = exp(-x.^2-y.^2);
```

Tworzenie obiektu: `F = scatteredInterpolant(x,y,z);`

```
Dokonanie interpolacji

[X,Y]=meshgrid(0:0.1:1,0:0.1:1);

Z=F(X,Y);
```

Temat 87
Aproksymacja – funkcja *polyfit*

Aproksymacja oznacza przybliżanie tzn. zastępowanie jednych wartości innymi, wygodniejszymi, z jakich względów. Matlab pozwala na aproksymację wielomianem.

Przykład:

```
p=polyfit(x,y,r); % x, y – serie danych, r – zadany stopień wielomianu przybliżającego
```

Ogólna postać wywołania funkcji polyfit
[p,s,mu] = polyfit(x,y,n)
p - wektor współczynników wielomianu
s - struktura opisująca błędy zawierająca:
R - czynnik dekompozycji QR macierzy Vandermonde'a
df - liczba stopni swobody
normr - norma wartości rezydualnych
mu - dwuelementowy wektor [mean(x), std(x)]

[y,delta] = polyval(p,x,S)

delta - błąd aproksymacji dla 95% przedziału ufności

NUMERYCZNE RÓŻNICZKOWANIE FUNKCJI

Temat 88
Numeryczne różniczkowanie funkcji

Przybliżoną wartość pochodnej można obliczyć przez obliczenie różnic pomiędzy wartościami tych samych współrzędnych sąsiadujących punktów funkcji zadanej numerycznie. Korzysta się tu z definicji pochodnej funkcji:

$$\frac{d\,f(x)}{dx} = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

Do wykonania tej operacji wykorzystuje się funkcję *diff*(), która oblicza różnice pomiędzy sąsiadującymi elementami wektora.

Przykład

```
>> dxdy = diff(y)./diff(x) % y jest wektorem z wartościami funkcji w punktach x
```

Temat 89
Inne funkcje związane z numerycznym obliczaniem pochodnej.

gradient	Numeryczne obliczenie gradientu funkcji
del2	Laplasian funkcji

Przykład

```
>> v = -2:0.2:2;
>> [x,y] = meshgrid(v);
>> z = x .* exp(-x.^2 - y.^2);
>> [px,py] = gradient(z,.2,.2);
>> contour(v,v,z)
>> hold on
>> quiver(v,v,px,py)
>> hold off
```

CAŁKOWANIE NUMERYCZNE

Temat 90
Całkowanie numeryczne

Problem polega na znalezieniu całki oznaczonej funkcji f(x) na przedziale <a,b>. Jest to łatwe, gdy znana jest funkcja pierwotna F(x) taka, że F(x)'=f(x) nie zawsze jest to jednak możliwe. Metody całkowania numerycznego (kwadratury) polegają na przybliżeniu funkcji podcałkowej f na danym przedziale <a,b> lub jego podprzedziałach przy pomocy innej funkcji, dla której wartość całki jest określona analitycznie. Matlab stosuje kwadratury Newtona-Cotesa - *quad* (interpolacja wielomianem drugiego stopnia) i Simpsona - *quadl* (interpolacja wielomianowa – dobierany stopień wielomianu).
Q=quad(f,a,b,tol,trace);
Q=quadl(f,a,b,tol,trace);
f – łańcuch zawierający nazwę funkcji, funkcja musi być umieszczona w odpowiednim skrypcie, musi zwracać wektor wartości a jej argumentem jest wektor elementów lub musi być zdefiniowana przy użyciu operatora @
a,b – przedział całkowania,
tol – wymagana tolerancja względna, domyślnie 10^(-3)
trace – parametr opcjonalny, pozwala na rysowanie wykresu z węzłami kwadratury.

Przykład 1

```
function [y] = funkcja_calkowana(x)
%%  funkcja
y=sin(x.*x);
%% Koniec
```

```
%%Wywołanie funkcji całkowania
>> q1 = quad('funkcja_calkowana',0,pi,1e-5,1);
>> ql = quadl('funkcja_calkowana',0,pi,1e-5,1);
```

Przykład 2:

```
>> fs = @(x) sin(x);
>> ql = quad1(fs,0,pi,1e-5,1);
```

W obecnej wersji matlaba rekomendowane jest używanie funkcji **integral**
q = integral(fun,xmin,xmax)
q = integral2(fun,xmin,xmax,ymin,ymax)
q = integral3(fun,xmin,xmax,ymin,ymax,zmin,zmax)

Temat 91
Inne funkcje związane z całkowaniem numerycznym funkcji

dblquad	obliczanie całek podwójnych
trapz	całkowanie metodą trapezów
triplequad	obliczanie całek potrójnych

Przykład:

```
>> F = @(x,y)y*sin(x)+x*cos(y);
>> Q = dblquad(F,pi,2*pi,0,pi);
>> X=[1:0.1:10];
>> Y=sin(X);
>> trapz(X,Y)
```

OBSŁUGA BŁĘDÓW

Temat 92
Komunikat błędu – funkcja *error*

Funkcja *error* wypisuje komunikat błędu: error('to jest błąd')
W celu wyprowadzenia na ekran komunikatu o błędzie w postaci okienka należy użyć funkcji *errordlg*(*'errorstrin','dlgname'*).

Przykład:

```
>> error('Cos jest nie tak');
>> errordlg('Cos jest nie tak','Komunikat o błędzie');
```

Temat 93
Komunikat ostrzeżenia – funkcja *warning*

Wypisuje komunikat ostrzeżenia.

Przykład:

```
>> warning('to jest ostrzeżenie');
```

Temat 94
Zmienne standardowe: *nargin*, *nargout*

Liczba parametrów w definicji funkcji może się różnić od liczby parametrów, jaką podaje użytkownik. Zmienna standardowa *nargin* określa dla danego wywołania funkcji z iloma parametrami funkcja ta została wywołana. Odpowiednio, zmienna *nargout* określa ile parametrów wyjściowych zostanie pobranych przez użytkownika.

Przykład:

```
function [y1, y2, y3, y4]=funkcja_x(x1, x2, x3, x4)
%%  [y1, y2, y3, y4]=funkcja_x(x1, x2, x3, x4)
% y1 ...y4 parametry wyjściowe funkcji
% x1....x4 parametry wejściowe funkcji

if (nargin<1), % nie ma wprowadzonych parametrów – wprowadzamy wartości domyślne
   x1=0; x2=0; x3=0; x4=0; % parametry wejściowe funkcji
elseif(nargin<2) % wprowadzono tylko jeden parametr
..... % jakaś akcja, np. komunikat błędu
elseif(nargin<3)
..... % jakaś akcja
end
% wszystkie warunki wprowadzania zostały sprawdzone

% testowanie ile parametrów wyjściowych chce otrzymać użytkownik, jeżeli użytkownik nie
chce dodatkowych parametrów wyjściowych szkoda czasu na ich obliczanie
if(nargout>1)
y2 = ....
y3= .....
y4= .....
% koniec funkcji
```

Zmienne *nargin*, *nargout* mogą być wykorzystywane do sprawdzania poprawności liczby wprowadzonych lub wyprowadzanych zmiennych z funkcji.

Przykład:

```
function funk(x,y)
if nargin ~= 2
    error('Wrong number of input arguments')
end
```

UWAGA: Do wprowadzania i wyprowadzania dowolnej liczby dowolnych elementów do i z funkcji wykorzystuje się odpowiednio funkcję *varargin* oraz funkcję *varargout*.

Temat 95

Reagowanie na nieprawidłowy argument funkcji

Do sprawdzenia czy dana zmienna ma oczekiwaną przez nas postać wykorzystujemy operatory typu is*.

ischar	Sprawdza czy dana wejściowa jest zmienną typu char
isempty	Sprawdza czy dana wejściowa jest zmienną pustą
isequal	Sprawdza czy dane 2 macierze są sobie równe
isglobal	Sprawdza czy dana wejściowa jest zmienną globalną
isinteger	Sprawdza czy wszystkie elementy ziennej są typu integer
islogical	Sprawdza czy dana wejściowa jest zmienną logiczną
isnan	Znajduje elementy nieskończone (NaN)
isnumeric	Sprawdza czy dana wejściowa jest zmienną numeryczną
isprime	Znajduje elementy będące liczbami pierwszymi
isreal	Sprawdza czy wszystkie elementy zmiennej są typu real
isscalar	Sprawdza czy dana wejściowa jest skalarem
issorted	Sprawdza czy dana wejściowa jest posortowana
issparse	Sprawdza czy dana wejściowa jest macierzą rzadką
isvector	Sprawdza czy dana wejściowa jest wektorem

Przykład:

```
if ~isreal(arg2),
   error('Zmienna musi być liczbą rzeczywistą')
end
```

```
>> a = 1:20; % zmienna wejściowa
>> p = isprime(a); % wektor z 1 tam gdzie dana liczba jest pierwsza
```

PRZETWARZANIE DŹWIĘKU

Przykładem sygnału 1 wymiarowego jest sygnał dźwiękowy.

Temat 96
Wczytanie pliku dźwiękowego – funkcja *audioread,* `[y,Fs] = audioread(filename)`

Do wczytania pliku dźwiękowego w formacie .wav wykorzystuje się funkcję *audioread*().

Przykład:

```
>> y = audioread('plik.wav'); % wczytanie pliku dźwiękowego
>> [y,Fs] = audioread('plik.wav'); % zwraca plik dźwiękowy y, częstość próbkowania Fs
```

Funkcja audioread czyta następujące formaty: wav, MP3, MPEG-4

UWAGA: Do wczytania ciągu danych niebędących plikiem dźwiękowym służy funkcja *load*.

Temat 97
Odtworzenie pliku dźwiękowego – funkcja *audioplayer*

Funkcją *audioplayer* można odtworzyć jako dźwięk dowolną zmienną będącą wektorem liczb rzeczywistych.
player = audioplayer(Y,Fs,nBits)
Alternatywnie używamy funkcji **play**

Przykład:

```
>> audioplayer(y); % odtworzenie pliku dźwiękowego y

>> d = rand(1,20000);
>> audioplayer(d); % odtworzenie wektora losowego

>> t = (0:0.001:1)';
>> y = sin(2*pi*50*t) + 2*sin(2*pi*120*t); sygnał o 2 składowych o częstościach 50 i 120 Hz
>> audioplayer(y) % odtworzenie dźwięku y
```

Temat 98
Zarejestrowanie dźwięku – funkcja *waudiorecorder*

Do zarejestrowania dźwięku wykorzystuje się funkcję *audiorecorder*. Funkcja ta może być wywoływane z kilkoma parametrami:
recorder = audiorecorder(Fs,nBits,NumChannels)

Przykład:

```
>> Fs = 44100 ;
>> nBits = 16 ;
>> nChannels = 2 ;
>> ID = -1; % default audio input device
>> recObj = audiorecorder(Fs,nBits,nChannels,ID);
>> recordblocking(recObj,5);
```

Inne funkcje związane z dźwiękiem:

`get`	Własności obiektu `audiorecorder`
`getaudiodata`	Pobiera zarejestrowane dane audio w postaci wektora danych
`getplayer`	Tworzy obiekt związane z `audioplayer`
`isrecording`	Sprawdza czy trwa nagrywanie dzwieku
`pause`	Zatrzymanie odtwarzania lub rejestracji dźwięku
`play`	Odtwarza dźwięk związany z obiektem `audiorecorder`
`record`	Tworzy obiekt związany z `audiorecorder`
`recordblocking`	Uruchamia rejestracje dźwięku
`resume`	Wznawia odtwarzanie lub rejestrację dźwięku
`set`	Ustawia własności obiektu `audiorecorder`
`stop`	Zatrzymuje odtwarzanie lub nagrywanie

Funkcja Obj=**audioDeviceReader** wyświetla informacje o karcie dźwiękowej, zaś getAudioDevices (Obj) - wyświetla dodatkowe informacje (np. o sterownikach).

Temat 99
Zapisanie pliku dźwiękowego – funkcja *audiowrite*

Do zapisania zarejestrowanego lub zmienionego pliku dźwiękowego wykorzystuje się funkcję audiowrite('nazwa pliku',zmienna z danymi, częstość próbkowania,)

Przykład:

```
>> audiowrite('muzyka.wav',y,11025); % zapisanie zmiennej y z próbkowaniem 11025 Hz w
                                     pliku muzyka.wav
```

Temat 100
Generowanie przebiegów czasowych

sawtooth	przebieg trójkątny (periodyczny)
square	przebieg prostokątny (periodyczny)
gauspuls	przebieg sinusoidalny modyfikowany Gaussem (nieperiodyczny)
chirp	Przebieg cosinusoidalny o zmiennym okresie

Temat 101
Fourierowska analiza próbek dźwiękowych – funkcja *specgram*
Funkcja *specgram* udostępniona jest w ramach *Signal Processing Toolbox*.
Zadaniem tej funkcji jest przeprowadzenie analizy częstotliwościowej sygnału zmiennego w czasie. Wynikiem działania funkcji jest obraz, na którego osi poziomej mamy informacje o czasie a na osi pionowej informacje o częstotliwościach.

Przykład:

```
>> specgram(y,512,2); % wyświetla
```

FOURIEROWSKA ANALIZA DANYCH

Analiza widma funkcji otrzymywanej za pomocą transformaty Fouriera stanowi podstawię większości algorytmów przetwarzania sygnałów. Matlab dostarcza wygodnych narzędzi do takiej analizy.

Temat 102
Jedno wymiarowa transformata Fouriera

Definicja ***fft(x)*** gdzie x jest wektorem próbek sygnału. ***ifft(x)*** – oznacza odwrotną transformatę Fouriera. Wszystkie pochodne funkcji transformaty Fouriera znajdują się w grupie *datafun*.

UWAGA: Funkcja *fft* i jej pochodne najszybciej wykonywana jest dla danych o rozmiarze będącym potęgą dwójki, np. 16, 32, 64, itd. Trochę wolniej trwają obliczenia dla danych o rozmiarze będącym liczbą pierwszą. Obliczenia dla danych o innych rozmiarach są zdecydowanie wolniejsze.

Temat 103
Inne pomocne funkcje

fftshift	przesunięcie odpowiednich elementów składowych sygnału
real	część rzeczywista liczby zespolonej
imag	część urojona liczby zespolonej
conj	sprzężenie zespolone
angle	część fazowa sygnału
abs	moduł sygnału
conv	splot dwóch wektorów
deconv	dekonvolucja dwóch wektorów
cov	kowariancja
decimate	przetworzenie wektora – rzadsze próbkowanie
interp	przetworzenie wektora – gęstsze próbkowanie
unwrap	Uciąglenie fazy

UWAGA: Aby obliczyć wartość natężenia sygnału należy skorzystać z następującego przekształcenia:

```
>> nat = abs(sygnal).^2;
```

Przykład:

```
>> Fs = 1000;                % częstotliwość próbkowania
>> T = 1/Fs;                 % czas próbkowania
>> L = 1000;                 % długość sygnału
>> t = (0:L-1)*T;            % wektor czasu
% Generacja przykładowego sygnału będącego suma dwóch sinusoidalnych przebiegów o
% częstotliwości 50 Hz oraz 120 Hz
>> x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
>> y = x + 2*randn(size(t));     % Dodanie szumu
>> plot(Fs*t(1:50),y(1:50))
```

```
>> title('Signal Corrupted with Zero-Mean Random Noise')
>> xlabel('time (milisekundy')
>> NFFT = 2^nextpow2(L); % Najbliższy wykładnik potęgi 2 dla wektora o długości L
>> Y = fft(y,NFFT)/L;
>>f = Fs/2*linspace(0,1,NFFT/2+1);
% Wykres amplitudy widma spektralnego.
>>plot(f,2*abs(Y(1:NFFT/2+1)))
>>title('Spektrum funkcji y(t)')
>>xlabel('Częstotliwość (Hz)')
>>ylabel('|Y(f)|')
>> % lub wykreśla widmo mocy
>> plot(f,(abs(Y(1:NFFT/2+1))).^2)
```

Temat 104
Dwu wymiarowa transformata Fouriera Transformata

Z przypadkiem dwu wymiarowej transformaty Fouriera najczęściej mamy do czynienia przy przetwarzaniu zdjęć.
Dwu wymiarowa transformata Fouriera – *fft2*.
Funkcje odwrotne: ifft2, ifftshift.(przesuwa zerowa częstotliwość do środka macierzy)

Przykład:

```
>> s = rand(128); % stworzenie 2 wymiarowego zbioru danych
>> fs = fftshift(fft2(s)); % 2 wymiarowa transformata Fouriera wraz z przesunięciem danych
```

Temat 105
Wielowymiarowa transformata Fouriera Transformata

Matlab oferuje możliwość obliczenia wielowymiarowej transformaty Fouriera przy wykorzystaniu funkcji *fftn*.

Przykład:

```
>> s4 = rand(32,32,32,32); % stworzenie 4 wymiarowego zbioru danych
>> fs4 = fftshift(fftn(s4));
```

Temat 106
Filtry analogowe i cyfrowe

Signal Processing Toolbox udostępnia wiele funkcji związanych z przetwarzaniem sygnałów za pomocą filtrów. Filtr analogowy, odpowiedź częstotliwościowa, realizowany jest przy wykorzystaniu funkcji *freqs*(*a, b, w*). Gdzie *a* i *b* są współczynnikami wielomianów a *w* jest wektorem z częstościami, dla których prowadzimy analizę.
Filtrację cyfrową, FIR (skończona odpowiedź impulsowe) lub IIR (nieskończona odpowiedź impulsowa), wykonuje się za pomocą funkcji *filter*(*b, a, x*). Gdzie *b* i *a* są współczynnikami odpowiednich wielomianów a *w* jest wektorem dyskretnych wartości sygnału na wejściu filtra.

Przykład:

```
>> data = [1:0.2:4]';
>> filter(ones(1,5)/5,1,data); % filtr cyfrowy tyu FIR (uśredniający)

>> a = [1 0.4 1]; b = [0.2 0.3 1]; w = logspace(-1,1);
>> freqs(b,a,w) % filtr analogowy
```

FALKOWA ANALIZA DANYCH

W przypadku gdy analizowane są dane (przebiegi czasowe) nie mogą być uznane za realizację procesu stacjonarnego losowego (tzn. ich własności statyczne nie zmieniają się w czasie) jak w przypadku analizy Fouriera wówczas dogodnym narzędziem są tzw. falki (wavelets). W matlabie dostępny jest toolbox (wavelts). Widmem mocy w tym przypadku jest funkcją czasu tzn. że możemy określić w którym memencie zmienia się charakterystyka częstotliwościowa badanego sygnału.

Transformata falkowa jest wykonywana za pośrednictwem funkcji **cwt.** Parametrami tej funkcji mogą być: 'morse', 'amor', i 'bump' odpowiadającej metodzie Morse, Morlet oraz Bump.

Przykład

```
>> Fs = 1000;              % częstotliwość próbkowania
>> T = 1/Fs;               % czas próbkowania
>> L = 1000;               % długość sygnału
>> t = (0:L-1)*T;          % wektor czasu
% Generacja przykładowego sygnału będącego suma dwóch sinusoidalnych przebiegów o
% częstotliwości 50 Hz oraz 120 Hz
>> x = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);
>>cwt(x,'morse',Fs)
```

PRZETWARZANIE VIDEO

Temat 107
Operacje wejścia wyjścia.

Do czytania plików video służy funkcja **VideoReader**, która tworzy obiekt typu video. Analogicznie funkcja **VideoWriter** umożliwia zapisania pliku na dysku. Do uzyskania informacji o pliku video stosuje się funkcję **mmfileinfo**.

Przykład:

```
>> v = VideoReader('myvideo.mp4');
```

Dodatkowe funkcje:

hasFrame	Sprawdza czy dostępna jest klatka video do wczytania z pliku
read	Czyta jedną lub więcej klatek filmu
readFrame	Czyta następna klatkę filmu
VideoReader.getFileFormats	Dostępne format video

Przykład:

```
>> v = VideoReader('myvideo.mp4');
>> frame = read(v,1);     %Read only the last video frame.
>> frame = read(v,Inf);   %Read frames 5 through 10.
>> frames = read(v,[5 10]); %Read from the 50th frame to the end of the video file.
>> frames = read(v,[50 Inf]);
```

Temat 108
Tworzenie animacji

Na podstawie wizualizacji danych w postaci wykresów możemy tworzyć animacje. Do tego celu służy m. in. Funkcja getframe, która zapisuje pojedynczy wykres jako klatka animacji

Przykład:

```
>> Z = peaks;
>> for j = 1:40
>>    X = sin(j*pi/10)*Z;
>>    surf(X,Z)
>>    drawnow
>>    F(j) = getframe(gcf);
>> end
>> movie(F)
```

Funkcja movie uruchamia animację

movie(F,n,fps) gdzie F- object frame, n –ilość powtórzeń (-1 oznacza animacje uruchomiona do przody I wstecznie), fps – prędkość w klatkach na sekundę

Pozostałe funkcję:

Im2frame - konwersja zdjęcia do obiektu frame

Fram2im - konwersja frame (jedna klatka) do zdjęcia

Getframe – zapisuje jako frame aktywne obiekt graficzny

Niektóre funkcje związane z przetwarzanie obrazów:

imread - czyta dane z pliku graficznego

imwrite – zapisuje dane do pliku graficznego

imfinfo - czyta parametry pliku graficznego

imhist – histogram obrazu

imresize – zmiana rozmiaru obrazu

imfilter – filtrowanie obrazu

imadjust – poprawa kontrastu obrazu

imshow – wizualizacja obrazu

rgb2gray – konwersja obrazu do obrazu w skali szarości

getpixelvalue – położenie kursora i wartości na obrazie

impixel – zwraca wartości RGB wskazane na obrazie za pomocą kursora

Dostęp do kamery (webcam)

webcamlist – informacje o zainstalowanych kamerach

cam=webcam – uchwyt do obiektu związanego z kamerą, własności obiektu cam mogą być zmieniane (np. Resolution, Exposure: -2, Brightness: 100)
preview(cam) – przechwyt strumienia z kamery z wizualizacja w oknie matlaba

closePreview(cam) – zamknięcie strumienia danych z kamery

img = snapshot(cam); - przechwycenie jednego obrazu z kamery

PODSTAWY STATYSTYKI W MATLABIE

Temat 109
Funkcje Statystyczne

Dostęp do opisu funkcji statystycznych: *help datafun*

max **[a,b]=max(A)** **max(A,dim)**	Element maksymalny **a** wartość maksymalna wektora **A** (lub kolejnych kolumn macierzy) lub, **b** indeks wartości maksymalnej wartość maksymalna względem wymiary dim macierzy A
min	Element minimalny
mean **mean(A,dim)** **nanmean**	Wartość średnia wartość maksymalna względem wymiaru dim ignoruje wartości NaN
median	Wartość medialna
prctile	Kwantyle
std	Odchylenie standardowe
var	Wariancja
sort	Sortowanie kolumn wg różnych wartości
sortrows	Sortowanie kolumn wg różnych wartości
sum	Sumowanie elementów
prod	Mnożenie elementów
hist	Histogram
histcounts	Histogram
cumsum	Zwraca wektor kolejnych skumulowanych sum elementów
cumprod	Zwraca wektor kolejnych skumulowanych iloczynów elementów
corrcoef	Współczynniki korelacji [R,P,RLO,RUP]=corrcoef(x,y,'alpha',0.01) P jest macierzą współczynników p – prawdopodobieństwo uzyskania współ. korelacji R jeśli rzeczywisty współczynnik korelacji jest zerowy. Jeśli p jest Male (np. <0.05) wówczas współ. korelacji jest wysoki RLO dolny i górny RUP przedział współ. korelacji dla poziomu istotności alpha (domyślna wartość 0.05)
cov	Macierz kowariancji
skewness	Skośność - 3 moment rozkładu
kurtosis	Kurioza (4 moment rozkładu gęstości prawodpodobienstwa). Dla rozkładu normalnego wynosi 3
geomean	Średnia geometryczna
moment(X,n)	Moment centralny n-tego rzędu

Okno statystyki- wybór z figure->tool->data statistics

Dopasowanie danych do funkcji gęstości prawdopodobieństwa:
Przykład:

```
pd = fitdist(x,'Normal')
```

Generowanie normalnego rozkładu prawdopodobieństwa:

```
y = normpdf(x,0,1);
```

Generowanie dystrybuanty
cdf

Generowanie różnych rozkładów gęstości prawdopodobieństwa:
pd=makedist('Lognormal')
pd=makedist('Lognormal','mu',5,'sigma',2)

INNE TEMATY

Funkcje do konwersji układów współrzędnych

[theta,rho] = cart2pol(x,y)
[theta,rho,z] = cart2pol(x,y,z)

[x,y] = pol2cart(theta,rho)
[x,y,z] = pol2cart(theta,rho,z)

[x,y,z] = sph2cart(azimuth,elevation,r)
[azimuth,elevation,r] = cart2sph(x,y,z)

Funkcje specialne
https://nl.mathworks.com/help/matlab/special-functions-1.html?s_tid=CRUX_lftnav

STRUKTURY DANYCH MATLABIE

Struktury w matlabie
Zmienne zgrupowane w bloku. Definiujemy je bezpośrednio za pomocą operatora”.” lub z wykorzystaniem funkcji (setfield). Struktura może zawierać zmienne dowolnych typów
Przykład:
data.name=’Testfile’
data.time=12.00
data.value=[12.1 13];
data=setfield(data,’x’,1);

inny sposób definiowania struktury
data = struct(field1,value1,fieldN,valueN)

Inne funkcje:

setfield - definiuje nowa zmienna w ramach struktury
getfield – odczytuje wartość zmiennej w ramach struktury
fieldnames – nazwy zmiennych w ramach struktury
isfield – sprawa czy zmiana jest strukturą
orderfields – ustawia alfabetyczną kolejność zmiennych w ramach struktury
rmfield – usuwa zmienna ze struktury
struct2cell – konwersja struktury do zmiennej komórkowej
properties – własności struktury
cell2struct – konwersja zmienne komórkowej do struktury

Wykonywanie operacji na strukturach
A = **structfun**(@mean,S)
Uwaga: jeśli struktura zawiera zmienną znakową operacje numeryczne na nie wykonywane są po konwersji do kody ASCII

Zmienne komórkowe (cell) mogą zawierać różne formaty danych np.:
C = {1,2,3;’text',rand(5,10,2),{11; 22; 33}}

cell2mat(C) – konwersja dla zmiennej komórkowej tego samego typu zmiennej podwójnej precyzji

`cell` | `cell2struct` | `cell2table` | `iscell` | `mat2cell` | `num2cell` | `struct2cell` | `table2cell`

Zmienne tabelaryczne
LastName = {'Sanchez';'Johnson';'Li';'Diaz';'Brown'};
Age = [38;43;38;40;49];
Smoker = logical([1;0;1;0;1]);
Height = [71;69;64;67;64];
Weight = [176;163;131;133;119];
BloodPressure = [124 93; 109 77; 125 83; 117 75; 122 80];

```
T = table(LastName,Age,Smoker,Height,Weight,BloodPressure)
Dodanie własnosci obiektu
T.Properties.Description = 'Patient data, including body mass index (BMI) calculated using Height and Weight';
Podsumowanie:
summary(T)
```