Politechnika Wrocławska

Wykresy

ale i funkcje

Wojciech Myszka

2024-10-17

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Funkcje

Po co?

1. Pojawiło się pytanie czy można zadać MATLABowi funkcję i poprosić, żeby narysował jej wykres.
2. No to trzeba wiedzieć coś o funkcjach...
3. Żeby wiedzieć coś więcej o funkcjach muszę powiedzieć coś o edytorze poleceń i skryptach...

Edytory

Nowy skrypt I

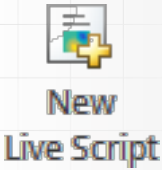
- Najprostsza metoda pracy to wpisywanie poleceń i oglądanie wyników.
- Gdy obliczenia polegają na wykonaniu kilku poleceń — można je wpisać do pliku tworząc „nowe polecenie".
- MATLAB wyposażony jest w edytor ułatwiający tworzenie i modyfikowanie takich plików z poleceniami (zwanych tradycyjne **skryptami**).
- Edytor pozwala na pracę z wieloma plikami, ma też możliwość edytowania jednego pliku w dwu oknach/miejscach.

Politechnika Wrocławska

Nowy skrypt II

- Dodatkowo ma możliwość odpluskwiania programów (debugowania — będzie o tym mowa później).
- Polecenia zapisuje się do pliku (o rozszerzeniu `.m`) i mogą być one uruchamiane na dwa sposoby:
  - naciskając klawisz Run (gdy plik otwarty jest w edytorze);
  - wpisując nazwę pliku jako polecenie MATLABa.
- Zapisując plik MATLAB może ostrzegać, że nazwa pliku pokrywa się z nazwą jakiejś jego funkcji — należy na to reagować.

New Live script



1. Drugim edytorem jest *Live editor*.
2. Pozwala on przeplatać obliczenia i ich wyniki z tekstem.
3. Wszystkie wykresy automatycznie umieszczane są pod poleceniami je generującymi.
4. Można za jego pomocą (podobnie jak w „zwykłym” edytorze) tworzyć skrypty.

Skrypty I

1. Po co są skrypty?
2. Mamy do wykonania zadanie (patrz na stronie zajęć)
   Rozwiąż układ równań $Mx = C$ mając macierz M współczynników niewiadomych oraz macierz C wyrazów wolnych

$$M = \begin{bmatrix} -2 & 0.5 & 4.2 & 8 \\ 0 & 4 & 8 & 2 \\ -5 & 7 & 3 & 1 \\ 10 & 12 & -6 & 4 \end{bmatrix} \quad C = \begin{bmatrix} 73.5 \\ 15.2 \\ -33 \\ 5 \end{bmatrix}$$

Zadanie jest dosyć proste:

- trzeba wpisać wartości do tablic
- wykonać prostą operację lewego dzielenia
- sprawdzić poprawność obliczeń

Skrypty II

```
M=[-2 0.5 4.2 8; 0 4 8 2; -5 7 3 1; 10 12 -6 4];
C=[73.5 15.2 -33 5]';
x=M\C
M*X-C
```

3. Gdy zdarzy się tak, że podczas wprowadzania danych do tablic popełnimy jakiś błąd — trzeba będzie go poprawić.
4. Edytor poleceń może nie być wystarczająco wygodny, zwłaszcza, gdy trzeba modyfikować ich wiele.
5. Znacznie lepsze będzie wpisanie poleceń do pliku; i uruchamianie ich z pliku
6. MATLAB oferuje dwa edytory

Skrypty III

6.1 „zwykły" (ikona opisana jako *New Script*)

Pliki zapisywane przez ten edytor powinny mieć rozszerzenie `.m` żeby MATLAB je rozpoznawał

6.2 „zaawansowany" (ikona opisana *New Live Script*)

pliki tworzone przez ten edytor muszą mieć rozszerzenie `.mlx` żeby matlab je rozpoznawał

7. Polecenia zapisane w pliku wykonać można na trzy sposoby:

7.1 Otwierając plik w edytorze i naciskając ikonkę *Run* (tak trójkącik)
7.2 Wydając polecenie zgodne z nazwą pliku: żeby wykonać polecenia zawartye w pliku `aa.m` piszemy po prostu `aa`
7.3 Wskazując plik w menedżerze plików MATLABa i wybierając polecenie *Run* z menu (lub naciskając klawisz F9)

Skrypty IV

W zasadzie nie ma wielkich różnic między użyciem poleceń zapisanych z użyciem tych edytorów.

Funkcje (ciąg dalszy)

Standardowe

czyli wszystkie dostępne w MATLABie i jego pakietach dodatkowych

Anonimowe I

- W oryginale nazywają się *anonymous* (stąd takie tłumaczenie)
- Czasami nazywane są *funkcjami lambda*

Funkcja anonimowa powiązana jest z obiektem zwanym *function handle* i używane są do zapisu bardzo prostych, jednolinijkowych wyrażeń.

Function handle może być **trochę** porównywane do znanego z C++ wskaźnika na funkcję...

Przykład

```
sqr = @(x) x.^2;
```

`sqr` to zmienna, która jest „wskaźnikiem" na funkcję o jednym argumencie `x` (`@(x)`) realizującą operację podniesienia do kwadratu.

Anonimowe II

Zwracam uwagę na tę kropkę: argumentem funkcji może być i skalar i wektor (i macierz)

Kolejny przykład funkcji, która korzysta z jakichś stałych:

```
a = -3.9;
b = 52;
c = 0;
parabola = @(x) a*x.^2 + b*x + c;
```

Gdy chcemy stworzyć funkcję, która nie ma parametrów

```
t = @() datestr(now);
```

Anonimowe III

(polecenie `now` odczytuje bieżący czas, funkcja `datestr` konwertuje wartość format czytelny)

Podczas używania tej funkcji mimo braku parametrów nawiasy są konieczne:

```
>> t()
ans =
    '18-Oct-2024 09:57:22'
```

Użytkownika: w pliku .m lub .mlx I

Kolejna możliwość zdefiniowania funkcji to zapisanie własnego jej algorytmu w pliku poleceń MATLABa. Możliwość ta znana jest również z innych języków programowania.

Definicja funkcji wygląda jakoś tak

```
function [y1,...,yN] = myfun(x1,...,xM)
...
end
```

W ogólnym przypadku (inaczej niż w C/C++) funkcja może zwracać wiele wartości.

- `x1` do `xM` to parametry wejściowe (argumenty) funkcji
- `y1` do `yN` to wartości **wyniki** (tablica!) zwracane przez funkcję
- `myfun` to nazwa funkcji

Użytkownika: w pliku `.m` lub `.mlx` II

- w miejscu kropeczek wpisujemy kod (algorytm) funkcji
- `end` (nieobowiązkowe) mówi gdzie się funkcja kończy

Funkcja zapisana musi być w pliku o nazwie `myfun.m` albo `myfun.mlx`

Przykładowe funkcje:

1. `addme()`
2. `kwadrat()`
3. `sześcian()`
4. `tryg()` zdefiniowana tak:

```
tryg = @(x) sin(x) + cos(x);
```

Wykresy

Najprostszy `plot()`

Już poznaliśmy.

- Wykres powstaje na podstawie dwu tablic danych zawierających współrzędne x i y punktów.
- Tablice muszą być jednakowej długości.
- Gdy jest tylko jedna tablica — zakłada się, że indeks tablicy jest współrzędną x
- Dodatkowe parametry pozwalają zdefiniować styl wykresu; można je zmienić korzystając z menu
- Wszystkie kolejne polecenia `plot()` wyświetlane będą w tym samym okienku — co czasami prowadzi do nieporozumień (zwłaszcza podczas korzystania ze skryptów)
- Trzeba pamiętać o użyciu poleceń:
  - `figure` (które tworzy dla każdego wykresu osobne okienko)
  - `hold` (które pozwala dodać coś do wykresu)

Politechnika Wrocławska

Inne rodzaje wykresów

Line Plots
plot
plot3
stairs
errorbar
area
stackedplot
loglog
semilogx
semilogy
fplot
fplot3
fimplicit

Scatter and Bubble Charts
scatter
scatter3
bubblechart
bubblechart3
swarmchart
swarmchart3
spy

Data Distribution Plots
histogram
histogram2
pie
pie3
scatterhistogram
swarmchart
swarmchart3
wordcloud
bubblecloud
heatmap
parallelplot
plotmatrix

Discrete Data Plots
bar
barh
bar3
bar3h
pareto
stem
stem3
stairs

Geographic Plots
geoplot
geoscatter
geobubble

Polar Plots
polarplot
polarhistogram
polarscatter
polarbubblechart
compass
ezpolar

Contour Plots
contour
contourf
contour3
contourslice
fcontour

Vector Fields
quiver
quiver3
feather

Surface and Mesh Plots
surf
surfc
surfl
ribbon
pcolor
fsurf
fimplicit3
mesh
meshc
meshz
waterfall
fmesh

Volume Visualization
streamline
streamslice
streamparticles
streamribbon
streamtube
coneplot
slice

Animation
animatedline
comet
comet3

Images
image
imagesc

Wykres funkcji I

No, ale pytanie było o wykres funkcji...

Żeby narysować wykres funkcji można użyć funkcji `fplot()`. Skoro już wiemy jak konstruować wykresy — możemy spróbować z funkcją `kwadrat()`

```
fplot(kwadrat)
```

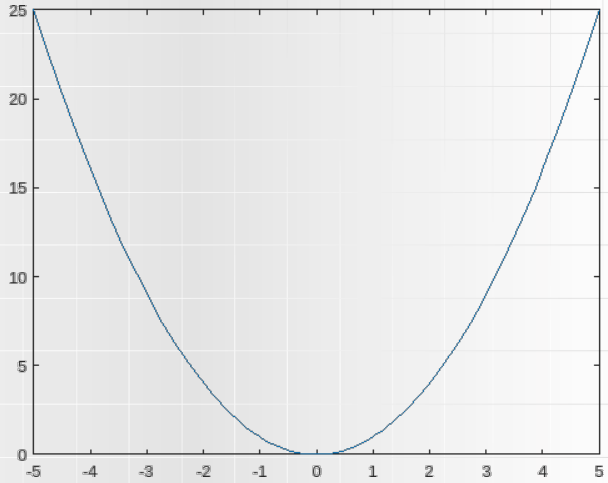
```
Not enough input arguments.
Error in kwadrat (line 3)
   y = x.^2;
```

Ale to nie działa...

Otóż argumentem funkcji `fplot()` musi być *function_handle*: wskaźnik na funkcję

Politechnika Wrocławska

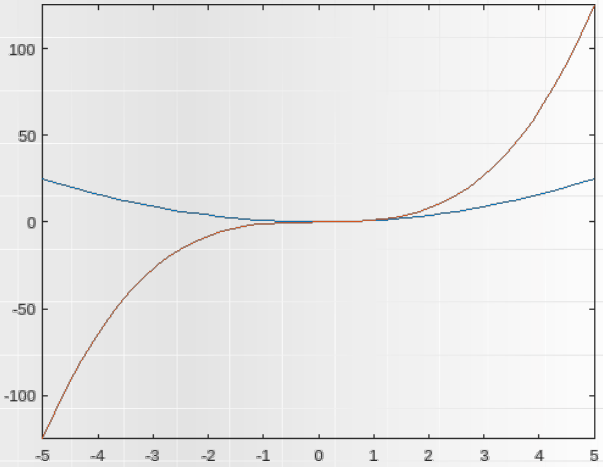
```
funkcja = @kwadrat;
fplot(funkcja)
hold on
```



Kilka wykresów

Zamiast używać kilku funkcji jako parametry do `fplot()` zalecam użyć polecenia `hold on` i `hold off`

```
funkcja1 = @szescian;
fplot(funkcja1)
```

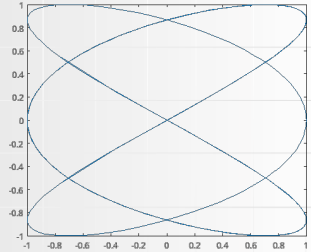


Użycie funkcji

1. Standardowy przedział w jakim rysowana jest funkcja to -5 do 5.
2. Można to zmienić, dodając drugi parametr: `fplot(f, xinterval)`, na przykład tak: `fplot(f, [-10,20])`
3. Funkcja `fplot()` może również służyć do rysowania wykresów funkcji zadanych parametrycznie, przez układ:

$$x = f_x(t) \qquad y = f_y(t)$$

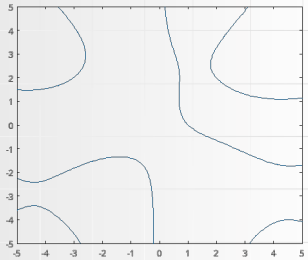
```
xt = @(t) cos(3*t);
yt = @(t) sin(2*t);
fplot(xt,yt)
```



Inne funkcje tego typu I

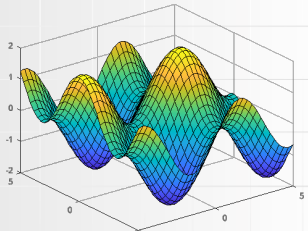
1. `fimplicit()` — rysuje wykres funkcji zadanej równaniem $f(x, y) = 0$

```
fp = fimplicit(@(x,y) y.*sin(x)
  + x.*cos(y) - 1)
```



2. `fsurf()` — rysuje wykres funkcji zadanej równaniem $z = f(x, y)$

```
fsurf(@(x,y) sin(x)+cos(y))
```

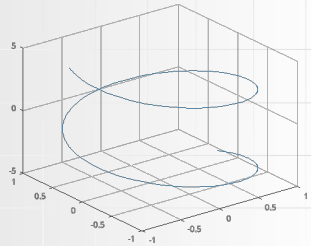


Inne funkcje tego typu II

3. `fplot3()` — rysuje wykres funkcji zadanej równaniami parametrycznymi

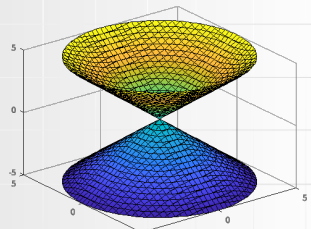
$$x = f_x(t) \quad y = f_y(t) \quad z = f_z(t)$$

```
xt = @(t) sin(t);
yt = @(t) cos(t);
zt = @(t) t;
fplot3(xt,yt,zt)
```



4. `fimplicit3()` — rysuje wykres funkcji zadanej równaniem $f(x, y, z) = 0$

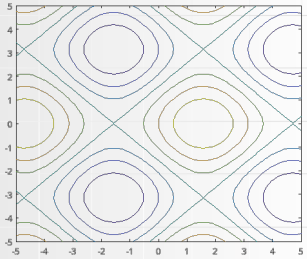
```
f = @(x,y,z) x.^2 + y.^2 - z.^2;
fimplicit3(f)
```



Inne funkcje tego typu III

5. `fcontour()` — rysuje wykres funkcji zadanej równaniem $z = f(x, y)$ dla różnych wartości z

```
f = @(x,y) sin(x) + cos(y);
fcontour(f)
```



Politechnika Wrocławska

Inne funkcje tego typu IV

Uwaga

- We wszystkich tych funkcjach standardowy zakres zmienności zmiennych to przedział $[-5; 5]$. Można to zmienić dodając odpowiedni parametr (najczęściej tablica).
- Również styl i kolor linii może być modyfikowany

Wygląd wykresu

1. `-`, `--`, `:`, `.-` — typ linii (ciągła, przerywana, kropkowana, kropka-kreska)
2. `o`, `+`, `*`, `.`, `x`, `_`, `|`, `s`, `d`, `^` `v`, `>`, `<`, `p`, `h` — znacznik (kółko, plusik, gwiazdka, kropka, krzyżyk, linia pozioma i pionowa, kwadrat, diament, trójkąt skierowany w różne strony, pięciokąt, sześciokąt)
3. `y`, `m`, `c`, `r`, `g`, `b`, `w`, `k` — kolor linii (yellow, magenta, cyan, red, green, blue, white, black) (ale jest więcej sposobów na modyfikowanie kolorów)

Gdy specyfikacja się nie pojawi, MATLAB będzie używał swoich ustawień standardowych

Zestawy danych będą wyświetlane na wspólnej dla wszystkich osiach liczbowych

Funkcja `figure()` I

1. Polecenie `figure` tworzy nowe okienko, w którym wyświetlany będzie wykres

   ```
   f = figure
   ```

   w zmiennej `f` znajdzie się „identyfikator” okienka, pozwalający dodawać (i modyfikować) jego zawartość

2. Polecenie może mieć szereg parametrów określających „cechy” okienka

   - nazwę (`Name`)
   - kolor (`Color`)
   - pozycję (`Position`)

   po nazwie parametru trzeba podać wartość cechy, na przykład

   ```
   figute('Name', 'Prosty wykres', 'Color', 'blue')
   ```

Funkcja figure() II

3. Gdy chcemy „wróćić” do wcześniej zdefiniowanego okna

```
f1 = figure % pierwsze okienko
...
f2 = figure % drugie okienko
...
figure(f1) % „wracamy" do pierwszego okienka
```

4. O tym poleceniu trzeba zwłaszcza pamiętać podczas wyświetlania wykresów ze skryptu
5. Gdy zagubimy się w wykresach polecenie `gcf` zwraca „identyfikator” aktualnego wykresu. (`gcf` — *get current figure*)

Politechnika Wrocławska

tiledlayout I

- Gdy chcemy na jednym obrazku przedstawić kilka wykresów obok siebie — możemy wybrać „układ kafelkowy”
- Nazywa to się *tiled layout*. Włączamy poleceniem
  - `tiledlayout(m,n)` — co oznacza siatkę `m` na `n` kafelków

    lub
  - `tiledlayout('flow')` — co powoduje dodawanie kolejnych wykresów, zmieniając w każdym kroku layout
- `nexttile` — te polecenie rozpoczyna każdy kafelek

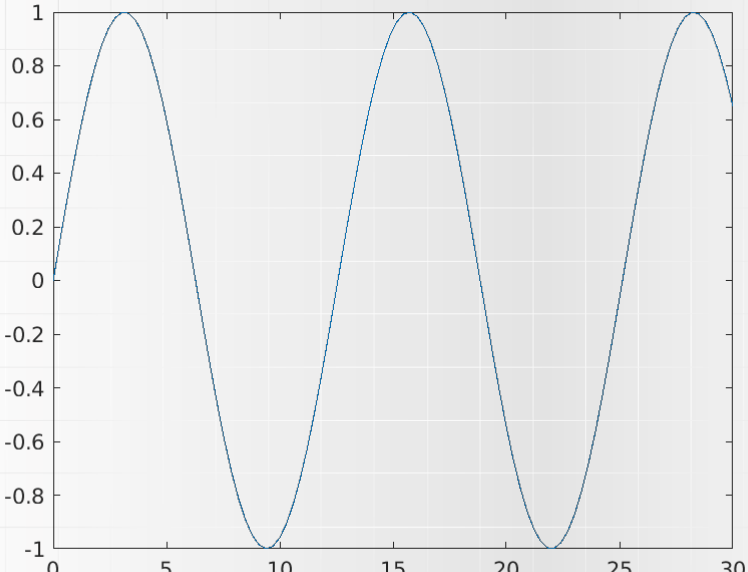
tiledlayout II

```
tiledlayout('flow')
nexttile
plot(...)
nexttile
plot(...)
nexttile
plot(...)
nexttile
```

da w efekcie

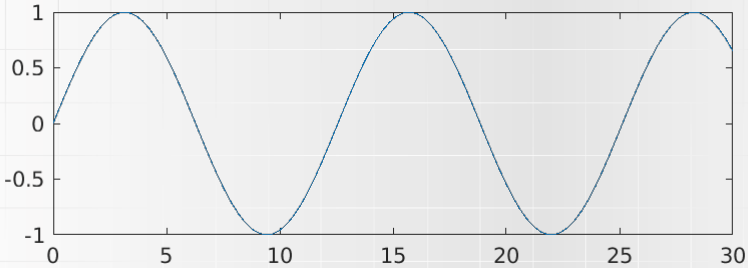
Politechnika Wrocławska

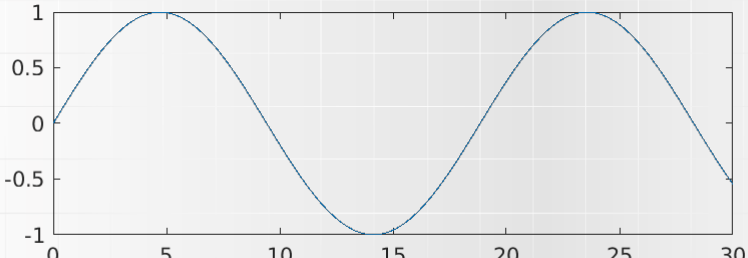
Przykład



Politechnika Wrocławska

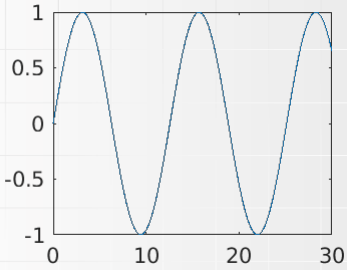
Przykład

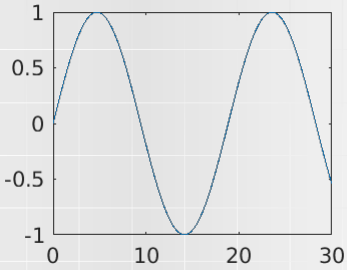


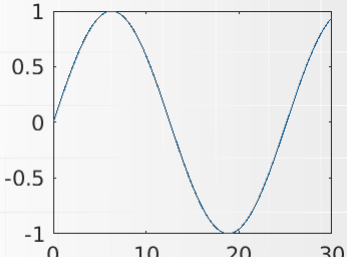


Politechnika Wrocławska

Przykład

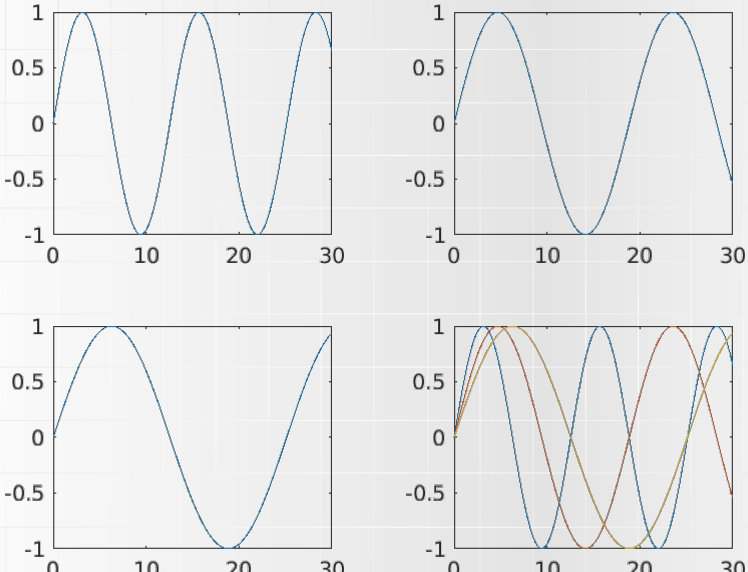






Politechnika Wrocławska

Przykład



Zapis wykresu do pliku I

1. Najprościej skorzystać z polecenia **File**→**Save** dostępnego w okienku wyświetlającym wykres
2. Zostanie on zapisany jako plik o rozszerzeniu `.fig` rozpoznwanym przez MATLABa; po kliknięciu w nie — ponownie otworzy się okienko z wykresem
3. Można też użyć polecenia **File**→ **Save as** pozwalającego zapisać wykres w różnych formatach
   - `fig`
   - `bmp`
   - `pdf`
   - `svg`
   - `jpg` (niezalecane)
   - `png`

Zapis wykresu do pliku II

- `eps`

4. Jest też funkcja `hgexport()`

```
f = figure
plot(...)
hgexport(f, `plik`)
```

Niestety, standardowo zapisuje w formacie `eps`. Możliwa jest zmiana tego formatu jak i innych parametrów tworzonego pliku, ale...

5. Kolejna funkcja to `saveas()` już nieco bardziej sensowna

```
f = figure
plot(...)
saveas(f,'plik')
```

Zapis wykresu do pliku III

rozszerzenie pliku powinno wskazywać na format w jakim ma być zachowany wykres

Extension	Resulting Format
`.fig`	MATLAB® FIG-file (invalid for Simulink block diagrams)
`.m`	MATLAB FIG-file and MATLAB code that opens figure (invalid for Simulink block diagrams)
`.jpg`	JPEG image
`.png`	Portable Network Graphics
`.eps`	EPS Level 3 Black and White
`.pdf`	Portable Document Format
`.bmp`	Windows® bitmap
`.emf`	Enhanced metafile
`.pbm`	Portable bitmap

Zapis wykresu do pliku IV

`.pcx`	Paintbrush 24-bit
`.pgm`	Portable Graymap
`.ppm`	Portable Pixmap
`.tif`	TIFF image, compressed

Zapis wykresu do pliku V

6. Jest wreszcie polecenie `exportgraphics()`, którego można użyć tak:

```
p = gcf;
exportgraphics(p,'r.png')
```

albo tak

```
p = figure
...
exportgraphics(p,'r.png')
```

Kończymy z funkcjami

Politechnika Wrocławska

Programowanie funkcji

1. Pisząc funkcję należy dobrze obmyślić jej
   - **algorytm**,
   - **listę argumentów** wejściowych

     i
   - **listę wyników**.
2. Wewnątrz funkcji można korzystać z następujących zmiennych:
   - `nargin` — liczba podanych (podczas wywołania funkcji) argumentów wejściowych;
   - `nargout`— liczba podanych (podczas wywołania funkcji) argumentów wyjściowych.

Politechnika Wrocławska

Dostępne polecenia

- `if`, `elseif`, `else`
- `switch`, `case`, `otherwise`
- `for`
- `while`
- `break`
- `return`
- `continue`
- `pause`
- `end`

Przykład nargin I

Tworzymy prostą funkcję sumującą dwa argumenty. Jej treść jest następująca:

```
function y = addme(a, b)
    y = a + b;
end
```

Zapisujemy ją w pliku `addme.m` i badamy zachowanie.

Zacznijmy od `help` (nawet jeżeli nie ma komentarza dostaniemy jakąś informację):

```
>> help addme
addme is a function.
    y = addme(a, b)
```

A teraz już użycie:

Przykład nargin II

```
>> addme(1)
Not enough input arguments.
Error in addme (line 2)
    y = a + b;
```

Dostajemy informację o zbyt małej liczbie argumentów.

```
>> addme(1, 2, 3)
Error using addme
Too many input arguments.
```

Teraz o zbyt wielkiej.

```
>> addme(1, 2)
ans =
     3
```

Przykład nargin III

Teraz OK.

Możemy jednak sprawdzić liczbę argumentów. Zmodyfikowana funkcja wygląda tak:

```
function y = addme(a, b)
    switch nargin
        case 2
            y = a + b;
        case 1
            y = a + a;
        otherwise
            y = 0;
    end
end
```

Przykład nargin IV

I kolejne testy:

```
>> addme(1)
ans =
     2
>> addme(2, 3)
ans =
     5
>> addme
ans =
     0
```

Poprawnie reaguje na sytuację braku, jednego i dwu argumentów.

Przykład nargin V

```
>> addme (2, 3, 4)
Error using addme
Too many input arguments.
```

W przypadku zbyt wielkiej liczby argumentów ciągle jest błąd!

Argumenty wyjściowe funkcji I

1. W odróżnieniu od wielu innych języków programowania funkcje MATLABa mogą zwracać kilka argumentów.

Niech nasza funkcja wygląda tak:

```
function [a b c] = abc(x, y)
    a = x + y;
    b = x - y;
    c = nargout;
end
```

Funkcja może zwracać do (bo nie musimy korzystać ze wszystkic!) trzech wartości:

Argumenty wyjściowe funkcji II

```
>> abc(1, 2)
ans =
     3
```

Jeden argument — tylko suma

```
>> [A B] = abc(1, 2)
A =
     3
B =
    -1
```

Dwa argumenty: suma i różnica

Argumenty wyjściowe funkcji III

```
>> [A B C] = abc(1, 2)
A =
     3
B =
    -1
C =
     3
```

Trzy argumenty — również liczba argumentów

```
>> [A B C D] = abc(1, 2)
Error using abc
Too many output arguments.
```

Argumenty wyjściowe funkcji IV

Cztery argumenty — błąd.

Zmienna `nargout` pozwala sterować obliczeniami w zależności od oczekiwań użytkownika.

Na przykład, gdy obliczenie jednej ze zwracanych wartości wymaga sporych obliczeń, można pominąć ten etap, gdy użytkownik nie potrzebuje jej.

Inne rodzaje wykresów

Rozkłady danych I

1. Histogramy
   - `histogram` Histogram plot
   - `histogram2` Bivariate histogram plot
   - `morebins` Increase number of histogram bins
   - `fewerbins` Decrease number of histogram bins
   - `histcounts` Histogram bin counts
   - `histcounts2` Bivariate histogram bin counts
2. `h - histogram(X)` rysuje histogram, parametry są dobierane automatycznie, choć można podać liczbę przedziałów, albo ich zakresy
3. `[N, edges] = histcount(X)` zwraca w formie wektorów wybrane automatycznie zakresy (`edges`) i liczbę obserwacji wpadających do każdego z nich (`N`); nie rysuje wykresu!

Rozkłady danych II

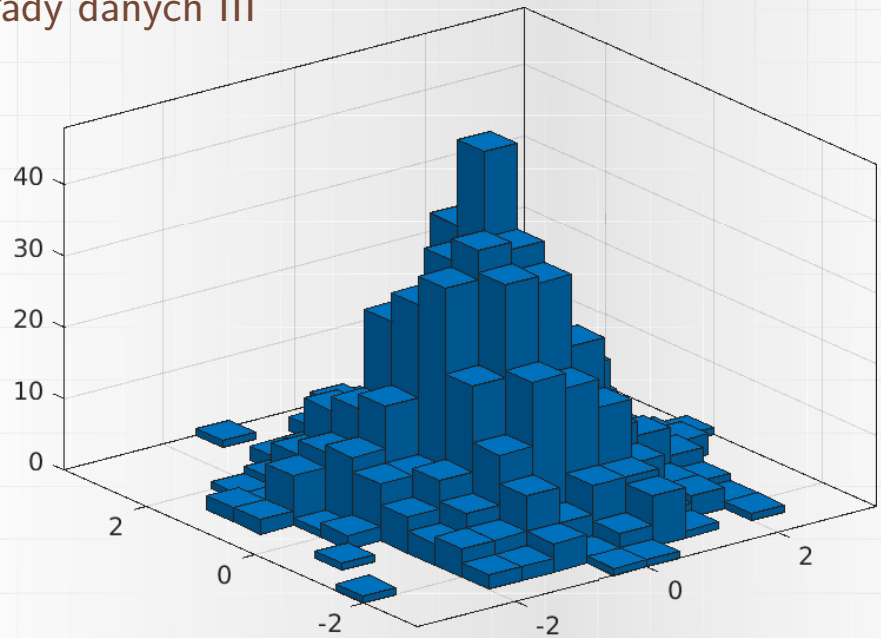
4. Mając narysowany histogram można odpytać o wszystkie ważne wartości

```
h = histogram(X);
...
h.NumBins     % liczba przedziałów
h.BinEdges    % granice przedziałów
h.Values % liczba zaobserwowanych wartości w każym przedzial
```

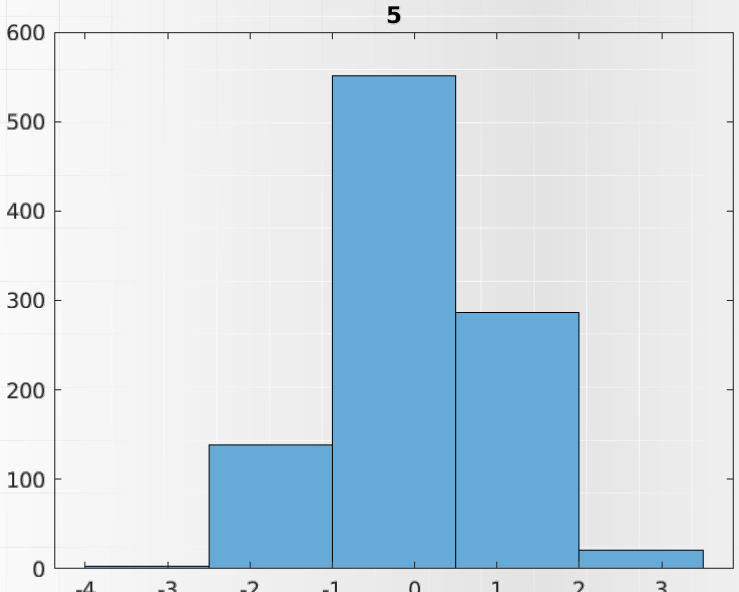
5. Gdy mamy dwie cechy, które chcemy przedstawić na wspólnym histogramie użyjemy funkcji `histogram2(X, Y)`.

Politechnika Wrocławska

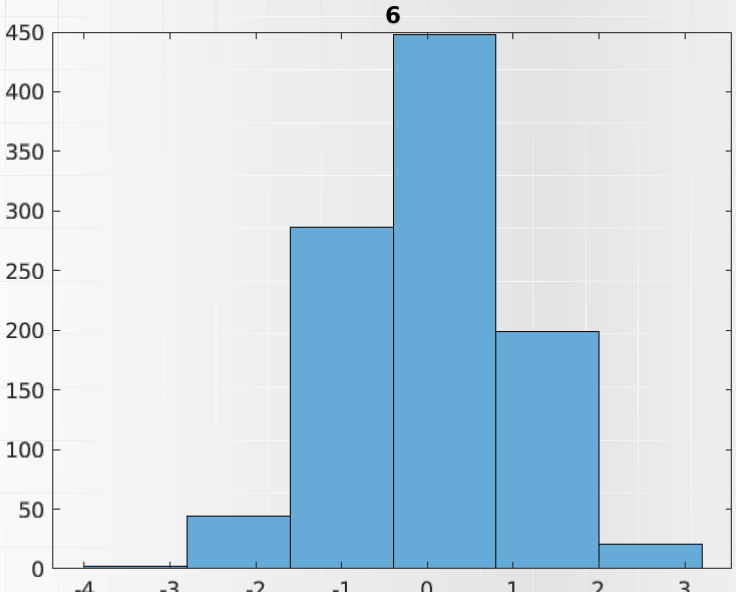
Rozkłady danych III



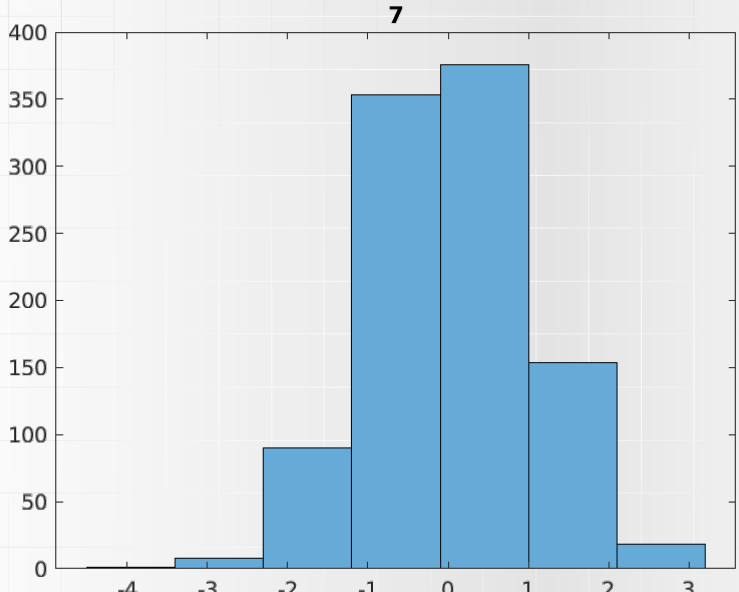
Przykład



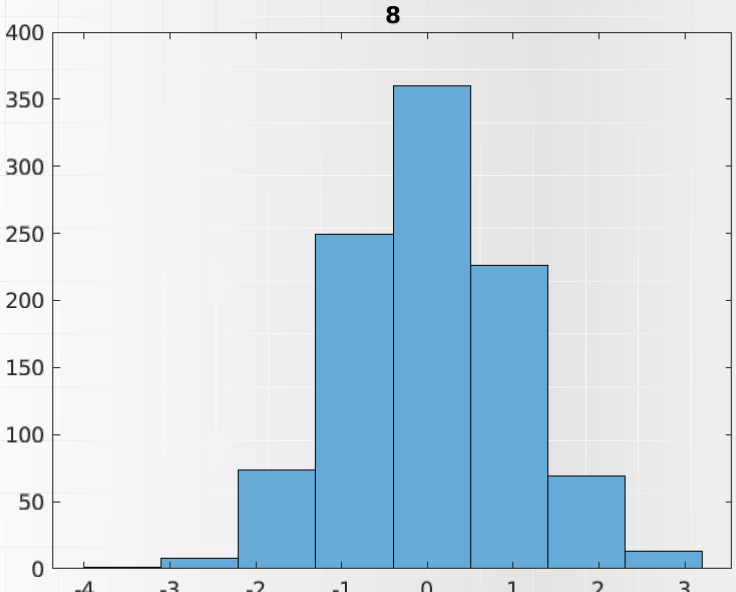
Przykład (morebins)



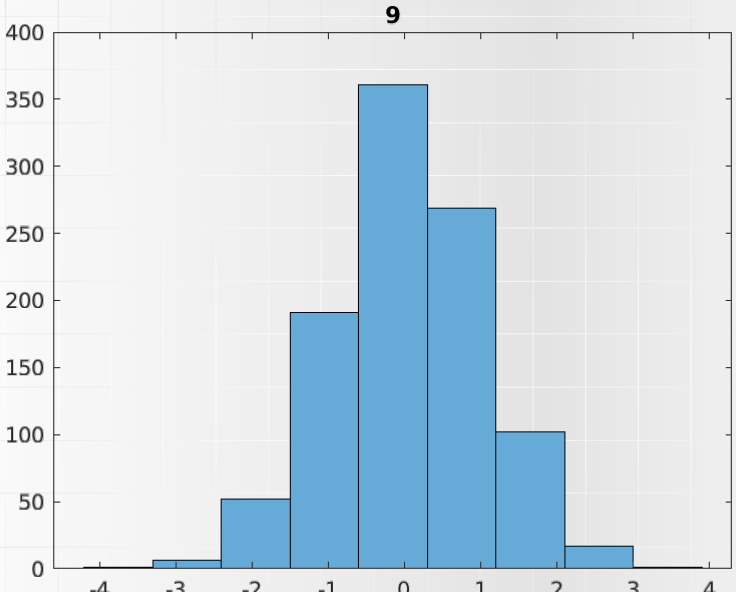
Przykład (morebins)



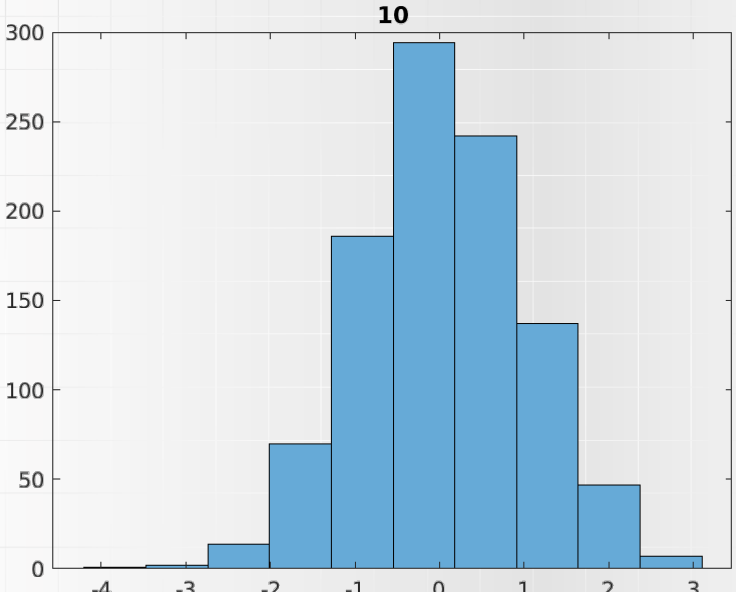
Przykład (morebins)



Przykład (morebins)



Przykład (morebins)



Przykład (automatyczny dobór przedziałów)

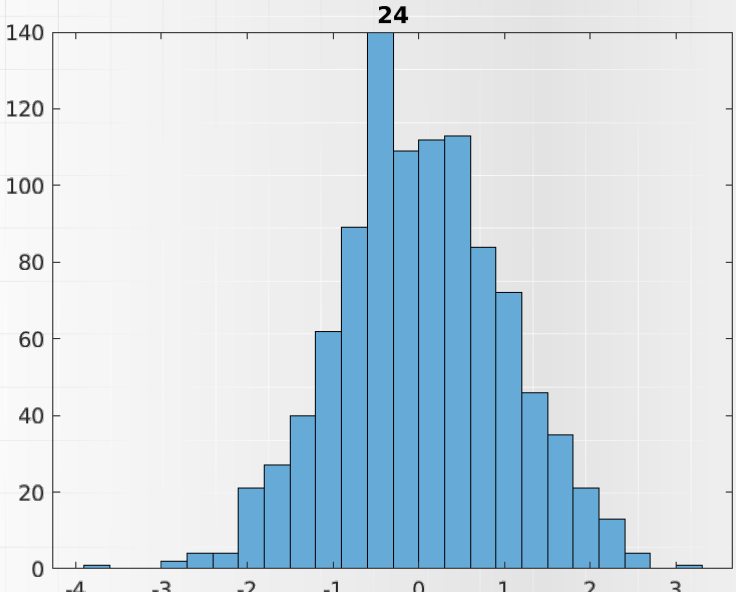


Diagram kołowy I

1. `pie(X)` rysuje diagram kołowy na podstawie danych w `X`. Każdy wycinek koła odpowiada jednej wartości z `X`
   - gdy `sum(X) <= 1` wartości odpowiadają powierzchni wycinków.
   - gdy `sum(X) < 1` — tylko część diagramu będzie wypełniona

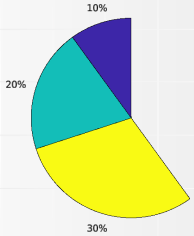
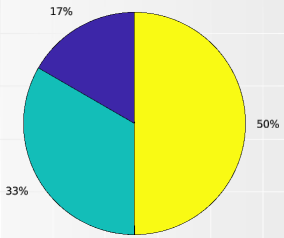


Diagram kołowy II

- gdy `sum(X) > 1` wartości są normalizowane `X/sum(X)` i rysowany jest wykres



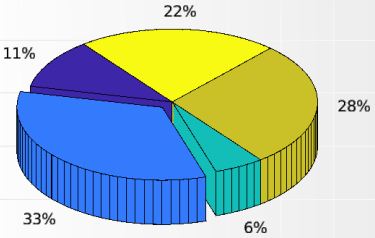
- Należy unikać tworzenia wykresów o zbyt wielu wycinkach

Politechnika Wrocławska

pie3

- diagram kołowy „3-d”

```
x = [1,3,0.5,2.5,2];
explode = [0,1,0,0,0];
pie3(x,explode)
```

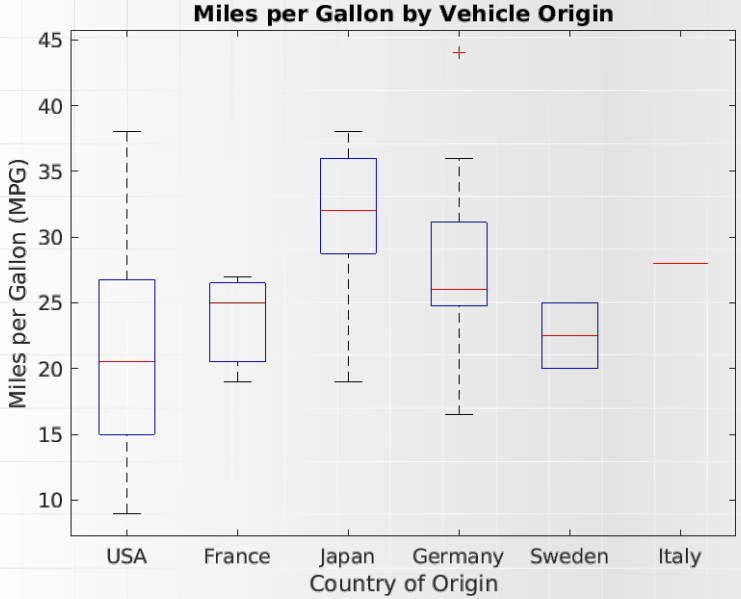


boxplot

Funkcja dokonuje podstawowej analizy statystycznej danych i przedstawia je w graficzny sposób:

- Na każdym polu środkowy znak wskazuje medianę,
- a dolna i górna krawędź pola wskazują odpowiednio 25. i 75. percentyl.
- „Wąsy" rozciągają się do najbardziej ekstremalnych punktów danych, które nie są uważane za wartości odstające,
- a wartości odstające są wykreślane indywidualnie za pomocą symbolu „+".

```
load carsmall
boxplot(MPG,Origin)
title('Miles per Gallon by Vehicle Origin')
xlabel('Country of Origin')
ylabel('Miles per Gallon (MPG)')
```


Miles per Gallon by Vehicle Origin
Miles per Gallon (MPG)
45
40
35
30
25
20
15
10
USA
France
Japan
Germany
Sweden
Italy
Country of Origin