MPDI2 - Informatyka

Wykład 11

- **Równania różniczkowe**
- **Przykłady zastosowań obliczeń symbolicznych**
- **Funkcje własne użytkownika**
- **Obliczenia statystyczne**
- **Rekurencja**

Rozwiązywanie równań różniczkowych

funkcja dsolve()

Jedna z funkcji w Matlabie obliczających symbolicznie rozwiązania **równań różniczkowych zwyczajnych (szukamy funkcji jednej zmiennej)**.

Rozwiązanie równania różniczkowego zwyczajnego polega na znalezieniu **funkcji** spełniającej równanie, które może zawierać:

- szukaną funkcję i jej pochodne
- inne funkcje standardowe

Przebieg obliczeń

Deklaracja funkcji symbolicznej, np. :

syms f(x)

Sformułowanie równania w postaci tożsamości, np.:

rownanie = diff(f) == cos(x)

$$\frac{df}{dx} = \cos x$$

albo inaczej:

rownanie = diff(f) - cos(x) == 0

Znalezienie rozwiązania:

rozwiazanie = **dsolve** (rownanie)

Przykład 1:

$$\frac{dy(t)}{dt} = 1 + y^2(t)$$

```
syms y(t)
f = dsolve( diff(y)== 1+y^2)
```

```
f =
        tan(C1+t)
```

Rozwiązanie zawiera stałą całkowania **C1**, czyli rozwiązań jest nieskończenie wiele.

Przykład 2: Równanie 3-go stopnia

$$\frac{\partial^3 u}{\partial x^3} = u$$

```
syms u(x)
rozw = dsolve( diff(u, 3) == u)
```

Wynik:

rozw =

C1*exp(x) + C2*exp(-x/2)*cos((3^(1/2)*x)/2) - C3*exp(-x/2)*sin((3^(1/2)*x)/2)

stałe całkowania: C1, C2, C3

Aby wyeliminować stałe całkowania należy zdefiniować warunki początkowe

$$\frac{\partial^3 u}{\partial x^3} = u$$

3 warunki początkowe

$$u(0) = 1, u'(0) = -1, u''(0) = \pi$$

```
syms u(x)
u1=diff(u) %definiujemy pochodną 1-go rzędu
u2=diff(u,2) %definiujemy pochodną 2-go rzędu
war = [u(0)==1, u1(0)==-1, u2(0)==pi]   %tablica warunków
u = dsolve(diff(u, 3) == u, war)
```

```
u =

    (pi*exp(x))/3 - exp(-x/2)*cos((3^(1/2)*x)/2)*(pi/3 - 1) - (3^(1/2)*exp(-x/2)*sin((3^(1/2)*x)/2)*(pi + 1))/3
```

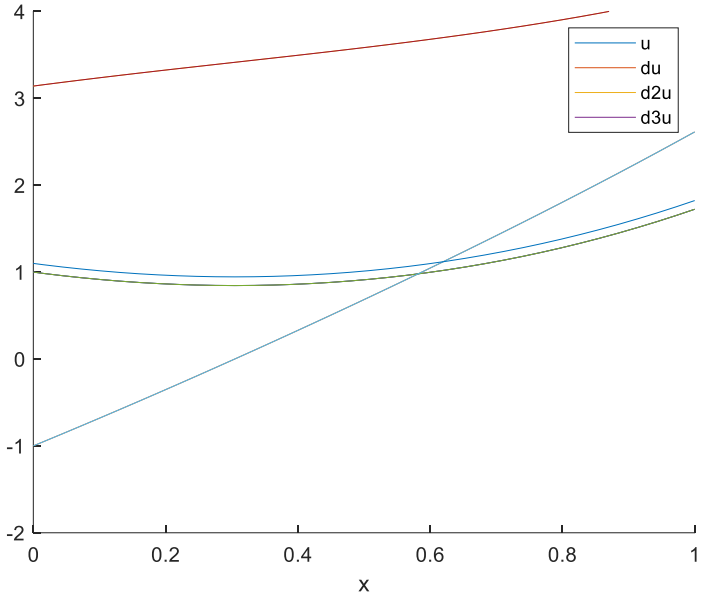
Możemy sprawdzić rozwiązanie:

```
spr=diff(u,3)-u
```

```
spr =
    0
```

Sprawdzenie rozwiązania na wykresach:

```
clc,clear
syms u(x)
hold on
u1=diff(u)
u2=diff(u,2)
u3=diff(u, 3)
war = [u(0)==1  u1(0)==-1  u2(0)==pi]
u = dsolve(u3 == u, war)
fplot(u)
fplot(diff(u))
fplot(diff(u,2))
fplot(diff(u,3)+0.1) % bo pokrywa się z u(t)
axis([0,1,-2,4])
legend('u', 'du', 'd2u', 'd3u')
```



Przykład 2

Równanie różniczkowe **drugiego stopnia** z dwoma warunkami początkowymi:

```
syms y(x)
dy = diff(y)
f = dsolve(diff(y,2) - cos(2*x)+y== 0, [y(0)==1,dy(0)==0])
f2 = simplify(f)   %uproszczenie
```

```
f =

(5*cos(x))/3 + sin(x)*(sin(3*x)/6 + sin(x)/2) -
(2*cos(x)*(6*tan(x/2)^2 - 3*tan(x/2)^4 + 1))/(3*(tan(x/2)^2 +
1)^3)

f2 =

    1 - (8*sin(x/2)^4)/3
```

Sprawdzenie rozwiązania:

```
syms x
s1= diff(diff(f2))-cos(2*x)+f2
s2=simplify(s1)  %uproszczenie
```

```
s1 =

 1 - 8*cos(x/2)^2*sin(x/2)^2 - cos(2*x)

s2 =

  0
```

Zestawienie poznanych funkcji Symbolic Toolbox

syms – deklaracja zmiennych symbolicznych

solve – symboliczne rozwiązywanie równań

subs – podstawianie danych do wyrażeń symbolicznych

diff - wyznaczanie pochodnych funkcji

int - wyznaczanie całek nieoznaczonych i oznaczonych

double – przekształcenie ułamka zwykłego do dziesiętnego

simplify – upraszczanie wyrażeń symbolicznych

ezplot lub fplot – wykres funkcji symbolicznej

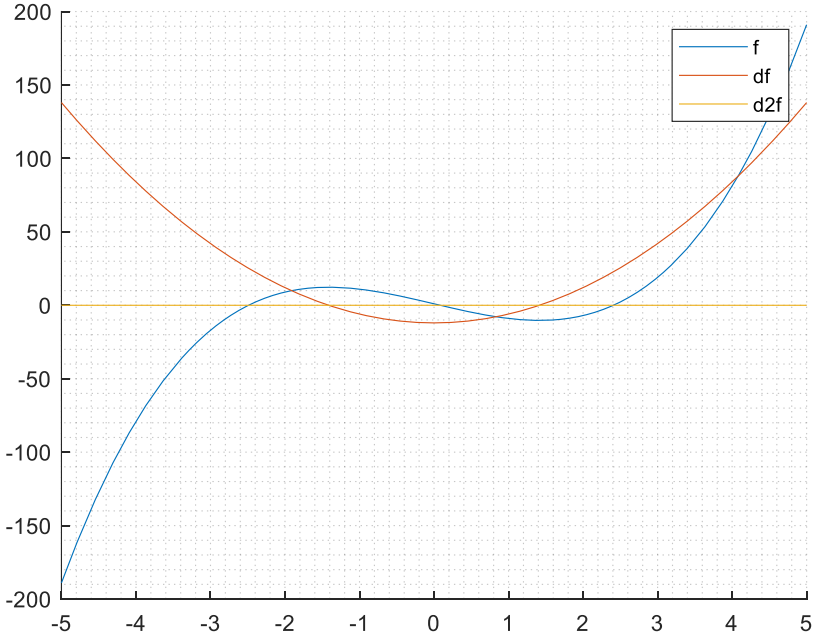
dsolve – rozwiązywanie równań różniczkowych

limit – granice ciągów i funkcji

Zastosowanie pochodnych dla badania przebiegu funkcji

```
clc, clear
syms x
f = 2*x^3-12*x+1;
hold on
fplot( [f  diff(f) 0] , [-5  5])
legend('f', 'df', 'd2f')
mzerowe=double(solve(f))                    %rozwiązanie funkcji
ekstrema=double(solve(diff(f)))             %rozwiązanie 1 pochodnej
pkty_przegiecia=double(solve(diff(f,2)))    %rozwiązanie 2 pochodnej
```

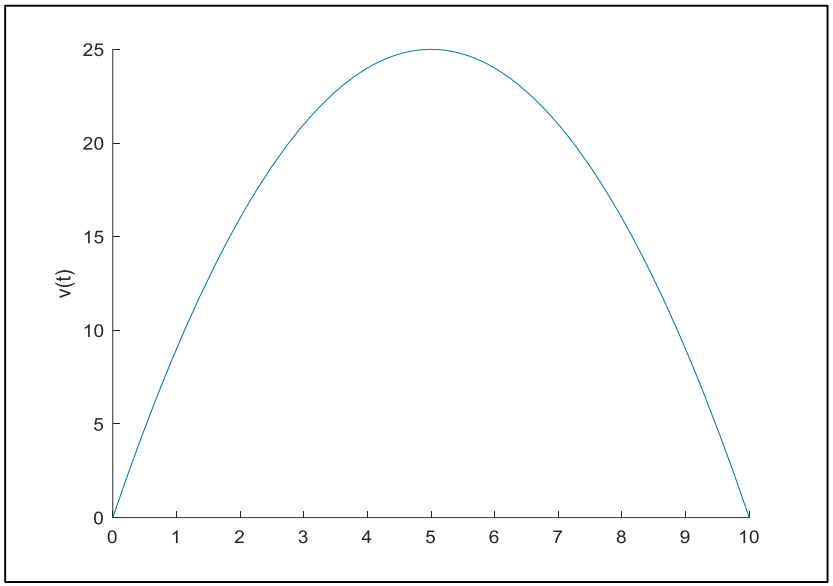
```
mzerowe =
    0.0834
    2.4067
   -2.4901
ekstrema =
    1.4142
   -1.4142
p_przegiecia =
     0
```

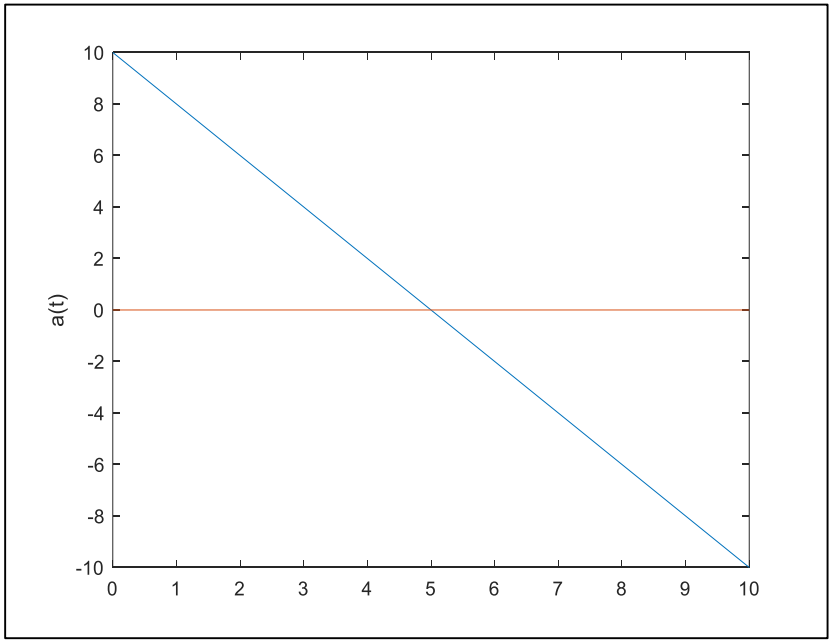


Równanie ruchu: $a = \frac{dv}{dt}$

Przykładowo dla: $\boldsymbol{v(t) = -t^2 + 10t}$

```
syms t
hold on
v=-t^2+10*t
figure(1) %okienko 1
fplot(v, [0 10])
ylabel('v(t)')  %opis osi y
a=diff(v) %pochodna prędkości
figure(2) %okienko 2
fplot([a 0], [0 10])
ylabel('a(t)')
```





Obliczanie pól powierzchni

```
clc, clear
syms x
f = sin(x)
pole= int(f, 0, pi) %pole powierzchni między krzywą a osią x w granicach (0, pi)
```

```
f =

    sin(x)

pole =

    2
```

Obliczanie pola powierzchni między krzywymi

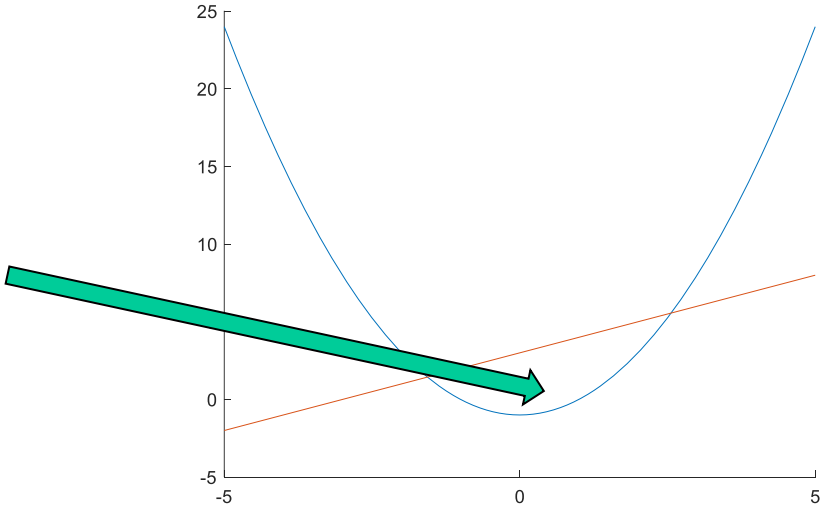
```
syms x
f1 = x^2-1
f2= x+3
hold on; fplot(f1);fplot(f2);                                   %wykresy
pkty_przec =double (solve (f1==f2))                             %granice całkowania
p1= double(int(f2-f1, pkty_przec(1), pkty_przec(2)))            %całka ozn. z różnicy funkcji
%albo różnica całek
p2= double(int(f2, pkty_przec(1), pkty_przec(2)) - int(f1, pkty_przec(1), pkty_przec(2)))
```

p1 =

11.6821

p2 =

11.6821



Definiowanie funkcji własnych użytkownika

Obliczenia przy pomocy tzw. <u>funkcji anonimowej</u>

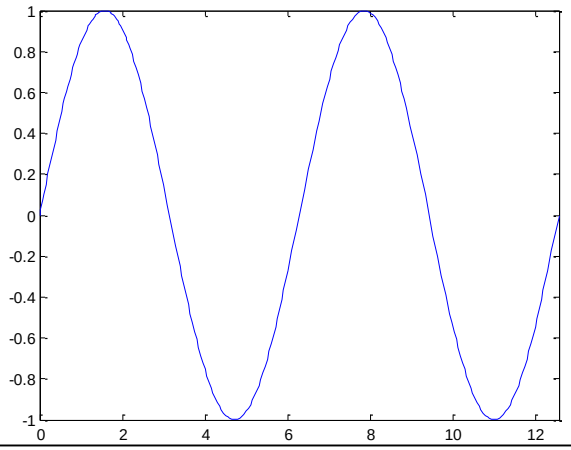
Definicja funkcji tworzona jest zazwyczaj w m-pliku i tylko w tym pliku wykorzystywana

Składnia funkcji anonimowej:

nazwa_funkcji=@(lista_argumentów) wyrażenie
nazwa_funkcji=@(lista_argumentów)[wyrażenia]

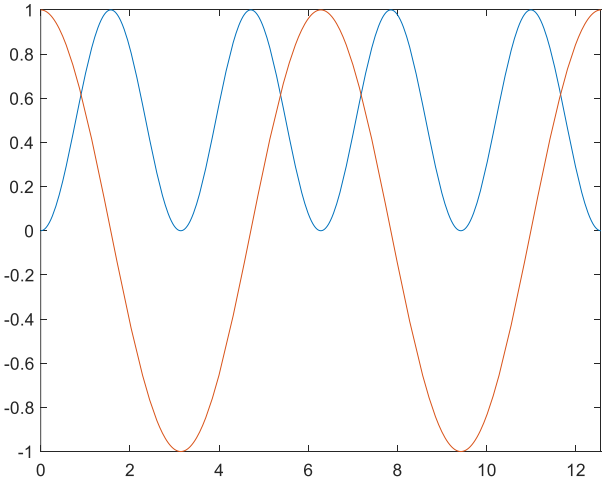
Przykład

```
sinusoida=@(t)sin(t)
fplot(sinusoida,[0,4*pi])
```



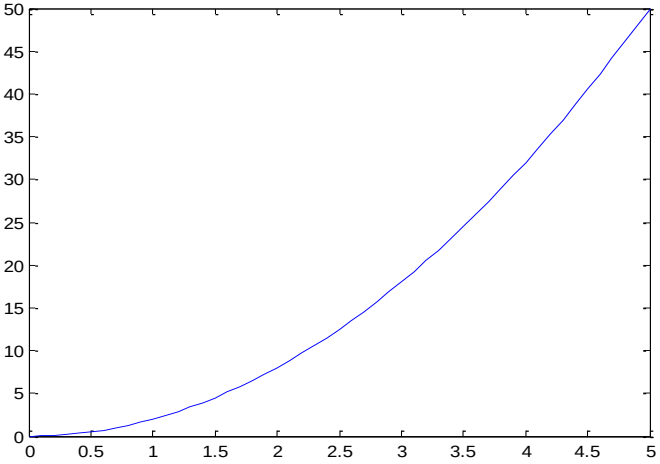

```
clc,clear
funkcje=@(t)[sin(t).^2 , cos(t)]   %definicja dwóch funkcji (dwa wyniki)
y=funkcje(pi/4)                    %wykorzystanie do obliczeń
fplot(funkcje,[0,4*pi])            %wykorzystanie do wykresów
```

```
@(t)[sin(t).^2,cos(t)]

y =

    0.5000    0.7071
```



Zadanie – funkcja ma obliczać drogę w ruchu jednostajnie przyspieszonym dla znanego czasu ruchu **t** i przyspieszenia **a**:

```
a=4;
droga = @(t)(a*t.^2/2);
t=0:0.1:5;
s=droga(t);
plot(t, s)
```



Funkcja anonimowa może mieć wiele argumentów

```
srednia=@(a,b)((a+b)/2)
x=srednia(4,6)
```

```
srednia =
    @(a,b)((a+b)/2)
x =
     5
```

```
ob_stozka=@(r,h)(pi*r*r*h/3)
x=ob_stozka(4,6)
```

```
ob_stozka =

    @(r,h)(pi*r*r*h/3)

x =

  100.5310
```

Funkcja anonimowa może też dawać wiele rezultatów

```
okrag=@(r)[2*pi*r,  pi*r^2] %TABLICA WYRAŻEŃ
x=okrag(4)
obwod=x(1)
pole=x(2)
```

```
obwod =
   25.1327

pole =
   50.2655
```

Definicja funkcji w osobnym m-pliku

Składnia definicji funkcji:

function [*wektor zmiennych rezultatów*]=nazwa(lista argumentów)

obliczenia rezultatów

end

Plik musi mieć nazwę jak nazwa funkcji
oraz
rozszerzenie **.m jak m-pliki**

Nasze zadanie …

Tworzymy <u>osobny</u> m-plik funkcyjny o nazwie droga.m:

```
function  y = droga(t,a)

y=a*t^2/2;

end
```

Teraz możemy wykorzystać funkcję w naszym m-pliku do budowania wektora w pętli for:

```
clc,clear
t=0:0.5:5; %wektor punktów czasu
n=length(t); %długość wektora
for i=1:n
        s (i)=droga(t(i),4);
end
plot(t,s)
```

Porównanie definicji funkcji w JavaScript i Matlabie

JavaScript

```
<HTML><HEAD>
<SCRIPT language="JavaScript">
function okrag(r)
  {
            A[0]=Math.PI*r*r
            A[1]=2*Math.PI*r
            return A
  }
</SCRIPT>
</HEAD><BODY>
<SCRIPT language="JavaScript">
promien=prompt("Podaj promień:")
wynik=okrag(promien)
document.write('Pole=',wynik[0],"<BR>")
document.write('Obwód=',wynik[1],"<BR>")
</SCRIPT>
</BODY> </HTML>
```

Matlab

plik okrag.m

```
function w=okrag(r)
   w(1)=pi*r*r;
   w(2)=2*pi*r;
```

plik nowy.m

```
clc, clear
pr=input('Podaj promień:');
wynik=okrag(pr);
fprintf('pole=%f\n',wynik(1))
fprintf('onwód=%f\n',wynik(2))
```

Wybrane funkcje statystyczne w Matlabie

Sumowanie elementów wektora:	**sum(A)**
Sortowanie elementów wektora:	**sort(A)** - rosnąco
	sort(A,'descend') - malejąco
Liczba elementów wektora:	**length(A)**
Średnia arytmetyczna:	**mean(A)**
Mediana:	**median(A)**
Odchylenie standardowe:	**std(A)** – dla próby
	std(A,1) – dla populacji
Wariancja:	**var(A)** – dla próby
	var(A,1) – dla populacji

Można sobie poradzić stosując iterację:

```
clc, clear
M=rand(1,100)
suma=0; %zerujemy sumę
for k=1:length(M)
        suma=suma+M(k); %sumowanie w pętli
end;
suma
srednia=suma/length(M)
%sprawdzenie- funkcje wbudowane
sum_s=sum(M)
sr_s=mean(M)
```

```
suma =
   52.7994
srednia =
    0.5280
sum_s =
   52.7994
sr_s =
    0.5280
```

Mediana

Mediana to wartość środkowa zbioru. Wartość mediany wskazuje, że połowa wyników ma wartość poniżej wartości mediany, a druga połowa ma wartość powyżej wartości mediany.

Po posortowaniu zbioru medianą jest:

- wartość elementu środkowego przy nieparzystej liczbie elementów w zbiorze
- średnia arytmetyczna dwóch środkowych elementów przy parzystej liczbie elementów w zbiorze.

```
clc, clear
M=rand(1,101);
M=sort(M)  %sortujemy rosnąco
% tworzymy własny algorytm
if rem(length(M),2)==1
   med= M((length(M)+1)/2)
else
  med= (M(length(M)/2)+M(length(M)/2+1))/2
end
%sprawdzenie algorytmu wykorzystując funkcję median
med_s= median(M)
```

```
med =

    0.5472

med_s =

    0.5472
```

Wariancja i odchylenie standardowe dla próby

Badamy **próbę**, czyli podzbiór pełnego zbioru (całej populacji)

Wariancja określa wielkość zróżnicowania wyników w zbiorze - czy różnice pomiędzy średnią a poszczególnymi wynikami są duże czy niewielkie.

$$w\_pr = \frac{\sum (x_i - \bar{x})^2}{N - 1}$$

Odchylenie standardowe próby:

$$\sigma\_pr = \sqrt{\frac{\sum (x_i - \bar{x})^2}{N - 1}}$$

Wariancja i odchylenie standardowe dla populacji

Jeśli badamy całą populację:

Wariancja populacji:

$$w\_pop = \frac{\sum(x_i - \bar{x})^2}{N}$$

Odchylenie standardowe populacji:

$$\sigma\_pop = \sqrt{\frac{\sum(x_i - \bar{x})^2}{N}}$$

Obliczenia wariancji populacji - zastosowanie iteracji

$$w\_pop = \frac{\sum (x_i - \bar{x})^2}{N}$$

```
N=100; % liczebność całej populacji
X=rand(1,N); %wylosujemy zbiór
% liczymy średnią arytmetyczną
srednia=mean(X);
sp=0;
% stosujemy wzór
for k=1:N
    sp=sp+(X(k)-srednia)^2;
end;
wariancja=sp/N %wariancja populacji
wariancja2=var(X,1) %sprawdzamy funkcję var
```

Rozkład normalny Gaussa - funkcja przybliżająca

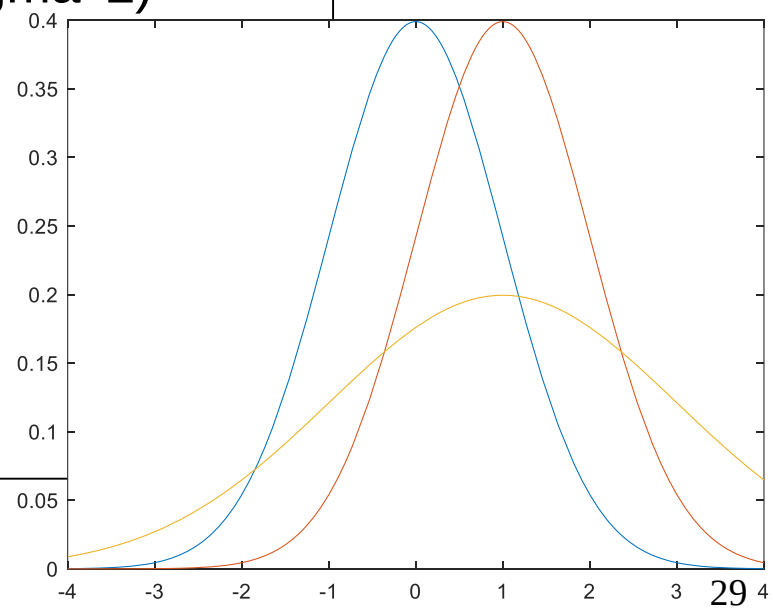
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

σ – odchylenie standardowe

μ - średnia arytmetyczna

Metoda symboliczna

```
clc
syms x sigma mi
f=1/sigma/sqrt(2*pi)*exp(-(x-mi)^2/2/sigma^2)
mi=0;sigma=1;
f1=subs(f) %podstawienie
mi=1; sigma=1;
f2=subs(f)
mi=1; sigma=2;
f3=subs(f)
fplot([f1 f2 f3],[-4 4])
```



Algorytmy rekurencyjne

Wiele problemów obliczeniowych można zdefiniować rekurencyjnie.

Rekurencja oznacza takie zdefiniowanie zagadnienia, gdzie w trakcie formułowania definicji **odwołujemy się do niej samej**.

Przykładem definicji rekurencyjnej może być zapis całkowitej, nieujemnej **potęgi rzędu *n* liczby rzeczywistej *x***:

$$x^n = \begin{cases} x^{n-1} * x & \text{dla } n > 0 \quad \text{(tu użycie definiowanej potęgi)} \\ 1 & \text{dla } n = 0 \end{cases}$$

Rekurencja w językach programowania jest realizowana za pomocą podprogramów **wywołujących kolejno same siebie** ze zmienianymi parametrami wywołania. Aby podprogramy rekurencyjne działały poprawnie **powinny zawierać warunek zakończenia rekurencji**, aby wywołanie wykonywane było skończoną liczbę razy.

Rekurencja daje proste programy lecz ma także wadę: każde wywołanie podprogramu wymaga wykonania przez procesor dodatkowych czynności, co spowalnia działanie programu oraz powoduje odłożenie na stos systemowy dużej liczby danych.

W Matlabie tworzymy definicję własnej funkcji:

```
function s=silnia (x)
% Funkcja wyznacza silnię liczby x
        if x==1
                s = 1;
        else
                s = silnia(x – 1)*x ;
        end ;
end
```

{w definicji funkcji wykorzystanie tej funkcji}

Najnowsze wersje Matlaba dopuszczają definiowanie rekurencyjne funkcji