6. Elementarne przykłady zastosowania

6.1 Badanie przebiegu funkcji

6.1.1 Wykresy funkcji (dwu- i trójwymiarowe)

Jedno z najprostszych zastosowań Matlaba i Scilaba to rysowanie wykresów funkcji zadanych w postaci wzoru. Dla ilustracji zagadnienia narysujmy rodzinę funkcji $y = ax^2$, dla trzech różnych wartości parametru a, którym odpowiadają trzy różne kolory i typy linii. Realizacja zadania została przedstawiona w dwóch wariantach - przy użyciu najprostszych komend oraz z elementami programowania:

```
x = [0:2:20];
plot(x, x.^2*1,'r-',x, x.^2*5,'g--',x, x.^2*6,'b.' );
xgrid(1);                          - lub set(gca(),'grid', [1,1]);
```

```
x = [0:2:20];
plot(x, x.^2*1,'r-',x, x.^2*5,'g--',x, x.^2*6,'b.' );
grid on
```

```
figure();set(gca(), 'auto_clear ', 'off '); xgrid(1);
a = [1 5 6];
x = [0:2:20];
typ=['r-','g--','b.'];             - tablica tekstów
for i=1:3, plot(x, x .^2*a(i), typ(i)); end
```

```
figure, hold on, grid on
a = [1 5 6];
x = [0:2:20];
typ=['r- ';'g--';'b. '];           - tablica 2-wymiarowa (spacje uzupełaniające)
for i=1:3, plot(x, x .^2.*a(i), typ(1,:)); end
```

Poza podstawowymi wykresami dwuwymiarowymi dostępne są również różne warianty wykresów trójwymiarowych. Przykład takiego wykresu przedstawiony poniżej przedstawia wykres funkcji $z = x \cdot e^{\left(-x^2 - y^2\right)}$, gdzie dziedzinę funkcji wyznaczają x=-2÷2 i y=-2÷3.

```
[x, y] = meshgrid(-2 : .2 : 2 , -2 : .2 : 3);
z = x .* exp(-x .^2 - y.^2);
surf(z);                           - lub mesh(z);
```

```
[x, y] = meshgrid(-2 : .2 : 2 , -2 : .2 : 3);
z = x .* exp(-x .^2 - y.^2);
surf(z);                           - lub mesh(z);
view(obrót_poziomy, obrót_pionowy);
```

Postać wykresu 3D zależy od wybranej funkcji graficznej i jest różna pod Matlabem i Scilabem.

6.1.2 Poszukiwanie miejsc zerowych i ekstremów funkcji

Podstawowe badania przebiegu zmienności funkcji obejmują wyznaczanie miejsc zerowych i ekstremów. Przykłady funkcji, które realizują najprostsze zadania dotyczące przypadku funkcji nieliniowej jednej zmiennej przedstawiono poniżej:

```
y = fsolve(x1, fnazwa)
```

```
fzero(badana_funkcja,  zakres_zmiennej_wejściowej)
fminbnd(badana_funkcja, zakres_zmiennej_wejściowej)
```

Dokładny sposób i przykłady zastosowania funkcji opisano w pomocy programów – poniżej dla ilustracji dwa proste wywołania:

```
xzero = fzero(@sin, 3)
xmin = fminbnd(@sin, 0, 2*pi)
```

Już na przykładzie tych dwóch funkcji widoczne są różnice, które dotyczą tego obszaru zastosowania Matlaba i Scilaba, począwszy od nazw funkcji i możliwości definiowania badanej funkcji.

6.2 Rozwiązywanie zagadnień algebraicznych

6.2.1 Definicja i operacje na wielomianach

Najprostszym sposobem definicji wielomianu jest wykorzystanie wektorów, to znaczy współczynniki wielomianu są zapisywane w wektorze, w którym pierwszy element odpowiada współczynnikowi przy najwyższej potędze zmiennej, na przykład:

- wielomian $x^2 + 2x + 5$ jest reprezentowany przez wektor `w1 = [1 0 0 2 5];`

Wektor współczynników wielomianu może być parametrem funkcji, na przykład do wyznaczenia pierwiastków wielomianu (w1):

```
p1=roots(w1)   → Scilab  1.0687573 + 1.2688874i      Matlab   1.0688 + 1.2689i
                         1.0687573 - 1.2688874i               1.0688 - 1.2689i
                       - 1.0687573 + 0.8212241i             -1.0688 + 0.8212i
                       - 1.0687573 - 0.8212241i             -1.0688 - 0.8212i
```

Kolejnym sposobem definicji wielomianu jest zapis symboliczny za pomocą zadeklarowanej wcześniej zmiennej symbolicznej (x):

```
x = poly(0, 'x');          - definicja zmiennej wielomianu (symbolu)
w2 = 1 + x + 2*x^2;        → 1 + x + 2x^2
```

Symboliczna postać wielomianu może być również używana jako parametr funkcji, obsługujących wielomiany, np.: `roots(w2)` .

Wielomian można również odtworzyć na podstawie jego pierwiastków, przekazanych do funkcji w postaci wektora, np.:

```
w3 = poly([1,2],'x')   → 2 - 3x + x^2
                       - wielomian w postaci symbolicznej
                       - sprawdzenie: roots(w3)  → 1  2
```

```
w3 = poly([1,2])   → 1   -3   2    czyli  x^2 - 3x + 2
                   - wielomian w postaci wektora
                   - sprawdzenie: roots(w3)  → 1  2
```

Inne funkcje - patrz przeglądarki pomocy na temat:

Polynomials | Matlab / Functions / Mathematics / Polynomials

6.2.2 Układy równań algebraicznych

Układy równań algebraicznych można zapisywać w sposób macierzowy, to znaczy układ m równań z n zmiennymi (x_1, ..., x_n) zapisany w postaci macierzowej $\mathbf{A}x = \mathbf{b}$ jest reprezentowany przez macierz A o wymiarach m x n, wektor kolumnowy b zawierający m wartości oraz wektor kolumnowy x reprezentujący zmienne układu. Wykonując operacje na macierzach można wyznaczyć wzór na poszukiwany wektor rozwiązań $x = \mathbf{A}^{-1}\mathbf{b}$ i zrealizować go różne sposoby:

- `x = A \ b` (zalecane rozwiązanie z zastosowaniem operatora lewostronnego dzielenia macierzy)
- `x = inv(A) * b`
- `x = A ^ (-1) * b`

6.3 Analiza danych

6.3.1 Interpolacja

Interpolacja polega na znalezieniu krzywej, która przechodzi przez zadane punkty nazywane węzłami interpolacji.

yi = interp1(xw, yw, xi);
yi = interp1(xw, yw, xi ,'*metoda*', [ekstrapolacja]);
gdzie: '*metoda*' = 'linear', 'spline', 'nearst'
	yi = interp1(xw, yw, xi);
	yi = interp1(xw, yw, xi ,'*metoda*', [ekstrapolacja]);
	gdzie: '*metoda*' = 'linear', 'spline', 'nearst', 'cubic'

Patrz też: interp2d, interp3d, interpln, interp	Patrz też: interp2, interp3, interp1q, pchip, interpft, interpn, spline

Dla ilustracji przykład interpolacji krzywej (wektory xi i yi) na podstawie kilku punktów funkcji sin (wektory xw i yw)

```
x=0:10; y=sin(x); xi=0:0.1:10;
yi = interp1(x, y, xi);
plot(x, y, 'o', xi, yi, '--')
```

```
x=0:10; y=sin(x); xi=0:0.1:10;
yi = interp1(x, y, xi);
plot(x, y, 'o', xi, yi, '--')
```

Możliwości powyższych funkcji w Matlabie i Scilabie różnią się w zakresie ekstrapolacji, czyli określania wartości funkcji poza znanym obszarem.

6.3.2 Aproksymacja

Aproksymacja oznacza przybliżanie, czyli zastąpienie jednych wartości innymi. Typowe zastosowanie to aproksymacja serii danych wielomianem wskazanego stopnia.

```
p = polyfit(x, y, stopien);
```

W przykładzie poniżej serię pomiarową (wektory x i y) wygenerowano jako fragment przebiegu sinusoidalnego, który próbowano aproksymować wielomianami drugiego i trzeciego stopnia – wynikiem jest wektor współczynników wielomianu (od najwyższej potęgi):

```
x=0:0.1:1; y=sin(x);
p2 = polyfit(x,y,2);
p3 = polyfit(x,y,3);
```

6.3.3 Całkowanie i różniczkowanie

Przykładem inne typowych operacji związanych z analizą danych jest całkowanie i różniczkowanie krzywych zadanych przez wektory wartości:

z = inttrap([x,] y); - *całkowanie metodą trapezów*

z = trapz([x,] y); - *całkowanie metodą trapezów*

y = diff(x [,n]); - *różnica kolejnych wartości (n-ilość powtórzeń)*

Poniżej wykonano aproksymację pola pod „połówką” sinusa i obliczono aproksymowany przebieg pochodnej tej krzywej:

X = 0:%pi/100:%pi; Y = sin(X);	- *„połówka” sinusa*	X = 0:pi/100:pi; Y = sin(X);	- *„połówka” sinusa*
Z = inttrap(X,Y)	→ Z = 1.9998355 (*pole pod krzywą*)	Z = trapz(X,Y)	→ Z = 1.9998 (*pole pod krzywą*)
X1 = diff(X);	- *wektor przyrostów zmiennej X*	X1 = diff(X);	- *przyrosty zmiennej X*
diff(Y)./diff(X)	- *aproksymacja pochodnej dY/dX*	diff(Y)./diff(X)	- *aproksymacja pochodnej dY/dX*

W powyższym przykładzie przebieg został określony wzorem, można więc zastosować alternatywny sposób wyznaczenia pola:

Z = integrate('sin(x)', 'x', 0, %pi) → Z = 2

6.4 Rozwiązywanie równań różniczkowych zwyczajnych

Symulacyjne rozwiązywanie równań różniczkowych to jedna z kluczowych funkcjonalności oprogramowania do obliczeń naukowo-inżynierskich. Dostępne są różne algorytmy (ang. solver) obliczania rozwiązania równań różniczkowych przy zadanych warunkach początkowych, różniące się dokładnością, szybkością działania, przeznaczeniem do określonego typu równań[1]. W funkcjach uruchamiających obliczenia poza równaniem różniczkowym i warunkami początkowymi oraz nazwą algorytmu podaje się zakres zmiennej niezależnej (często jest to czas *t*) i parametry algorytmu:

x = ode(**x0**, **t0**, **t**, **funkcja**) [**x**, w, iw]=ode([type], **x0**, **t0**, **t** [,rtol [,atol]], **funkcja** [,jac] [,w,iw])	[t,**x**] = *solver* (**'funkcja'**, [**t0** tf], **x0**) [t,**x**] = *solver* (**'funkcja'**, [**t0** tf], **x0**, options, p1, p2, ...)

Podstawowe parametry funkcji:

x – wynik obliczeń, **x0** – warunki początkowe, **t0** – czas początkowy, **funkcja** – nazwa pliku z równaniami różniczkowymi

t – wektor czasu	**[t0 tf]** – wektor z czasem początkowym i końcowym
Wybór solvera przez parametr type : 'lsoda' (domyślnie) 'adams', 'stiff', 'rk', 'rkf', 'fix', 'discrete', 'roots' Opcje solvera zawane w parametrach funkcji: rtol, atol	Wybór solvera przez nazwę funkcji *solver*: ode45 (najczęściej) ode23, ode113, ode15s, ode23s, ode23t, ode23tb, Opcje solvera options definiowane za pomocą funkcji, np.: options = odeset('RelTol', 1e-4, 'AbsTol', [1e-4 1e-4 1e-5]);

Równania różniczkowe, które solvery potrafią rozwiązać mogą być liniowe lub nieliniowe ale muszę to być równania pierwszego rzędu lub układy równań pierwszego rzędu [1][7]. Równania różniczkowe należy zapisać w pliku funkcyjnym.

1. Przykład równania pierwszego rzędu: $\dot{x}(t) = \sin(t * x(t)),\ x(0) = 0.2$

```
function [wynik]=sin_rownanie(t, x)
   wynik = sin(t*x);
endfunction
exec('sin_rownanie.sci');          //wczytanie funkcji
x0 = 0.2 ;  t = 0:0.1:15;          //war.początkowe i wektor czasu (50s)
x= ode(x0, 0, t, sin_rownanie);
plot(t, x)
```

```
function[wynik] = sin_rownanie(t, x)
 wynik = sin(t*x);


x0 = 0.2;  tend = 15;          %war.początkowe i czas trwania
[t,x] = ode45('sin_rownanie',[0 tend],x0);
plot(t, x)
```

2. Przykład równania drugiego rzędu: $\ddot{x}(t) - c\left(1 - x^2(t)\right)\dot{x}(t) + x(t) = 0 \Rightarrow \begin{bmatrix} x = x_1 & \leftarrow x \\ \dot{x}_1 = x_2 & \leftarrow \dot{x} \\ \dot{x}_2 & \leftarrow \ddot{x} \end{bmatrix} \Rightarrow \begin{cases} \dot{x}_1(t) = x_2(t) \\ \dot{x}_2(t) = c\left(1 - x_1^2(t)\right)x_2(t) - x_1(t) \end{cases}$

```
function [xprim]=vdp_rownanie(t, x)
  global c;                        //działa też bez globlal ale nie polecane
  xprim(1) = x(2);
  xprim(2)=c* (1 - x(1)^2) * x(2) - x(1);
endfunction
global c
c=10;
x0 = [-2.5; 2.5];  t =0 :.01:50;   //war.początkowe i wektor czasu (50s)
exec('vdp_rownanie.sci');          //wczytanie funkcji
[x] = ode(x0, 0, t, vdp_rownanie);
plot( t, x(1,:),'r' )
```

```
function[xprim] = vdp_rownanie(t, x)
 global c;                         %koniecznie global
 xprim = [x(2); c* (1 - x(1)^2) * x(2) - x(1)];


global c
c=10;
x0 = [-2.5; 2.5]; tend=50;         %war.początkowe i czas trwania = 50 sek

[t, x] = ode45('vdp_rownanie',[0 tend],x0);
plot( t,x(:,1),'r' );
```

[1] np. równania „zwykłe" (nonstiff) i znacznie trudniejsze do obliczania równania sztywne (stiff)