MATLAB dla o(d)pornych
(wersja robocza)

Maciej Krawiecki

12 października 2003

Przedmowa

Wyłącznym celem przyświecającym autorowi było zebranie w możliwie krótkim tekście opisów i przykładów zastosowania najniezbędniejszych poleceń programu MATLAB w takim zakresie, w jakim jest to niezbędne studentom odrabiającym Laboratorium Metod Numerycznych w Zakładzie Teorii Sterowania Instytutu Automatyki Politechniki Łódzkiej. Dlatego w poniższym opracowaniu omówiono niewielki podzbiór poleceń pakietu MATLAB, a studenci zainteresowani pełnymi możliwościami programu lub jego rozszerzeń, powinni skorzystać z podanej literatury. Rozmyślnie pominięto opis większości najnowszych rozszerzeń języka o nowe typy danych czy o elementy programowania obiektowego. Nie omawiano też pakietów procedur (toolboxów), w szczególności SIMULINK'a. W rezultacie, z bardzo niewielkimi wyjątkami, wszystkie poruszone zagadnienia odnoszą się zarówno do wersji 4.2b, jak i 5.X. Zasygnalizowano także niektóre zmiany, wprowadzone w wersji 6.0.

Niniejszy tekst jest we wczesnej fazie rozwojowej, dlatego autor prosi czytelników o wyrozumiałość i sygnalizowanie wszelkich zauważonych pomyłek.

Rozdział 1

Historia i współczesność MATLAB'a

Pierwszą wersję programu MATLAB napisał ok. 1980 r. w FORTRAN'ie prof. Clive Moler, aby ułatwić sobie obliczenia z dziedziny algebry liniowej. Stąd wywodzi się nazwa programu „MATLAB", będąca akronimem słów Matrix Laboratory. Ta pierwsza wersja była interpreterem języka poleceń pozwalającego zapisywać obliczenia na wektorach i macierzach w formie zbliżonej do notacji matematycznej i dającym dostęp do procedur numerycznych z bibliotek LINPACK oraz EISPACK. Zainteresowanie i pozytywna reakcja wśród znajomych (prof. Moler rozdał kilkaset taśm z kodem źródłowym) oraz rozpowszechnienie komputerów osobistych (PC) skłoniły autora MATLAB'a do komercjalizacji projektu: wraz z Johnem Little założyli firmę MathWorks Inc. i stworzyli PC–MATLAB przepisując kod programu w języku C oraz wzbogacając o funkcje graficzne. Ten krok (przeniesienie MATLAB'a na platformę PC) był przypuszczalnie najważniejszą decyzją strategiczną, która dała MATLAB'owi pierwsze miejsce wśród pakietów do obliczeń naukowo-technicznych. Aż do wersji 3.5 MATLAB był programem pracującym pod kontrolą MS–DOS lub PC–DOS. Werja 3.5 przeniesiona została do środowiska Windows i na Macintosh'a. Wersja 4.0 wprowadziła macierze rzadkie (ang. „sparse"), znaczne rozszerzenia funkcji graficznych (tzw. Handle Graphics) oraz wersje na platformy unixowe. Obok rozpowszechnienia spowodowanego dostępnością na platformy PC i Macintosh do sukcesu MATLAB'a przyczyniła się polityka licencyjna polegająca na udzielaniu znacznych zniżek wyższym uczelniom. Pozwoliło to wygrać z innymi podobnymi pakietami jak CTRL–C czy Matrix–X. Wersja 5.0 przyniosła dalsze udoskonalenie funkcji graficznych i bardzo znaczne zmiany języka polegające na wprowadzeniu nowych typów danych i programowania obiektowego z możliwością przeciążania operatorów włącznie. Przez lata liczba rozszerzeń MATLAB'a w postaci pakietów funkcji,

tzw. toolboxów stała się tak wielka, a zakres ich zastosowań tak szeroki, że nie jest możliwe omówienie ich wszystkich w jednej, choćby bardzo grubej, książce. Oferowane przez firmę MathWorks oraz niezależnych producentów toolboxy obejmują specyficzne zagadnienia teorii sterowania (kilkanaście pakietów), sieci neuronowe, statystykę, obróbkę obrazów, rozwiązywanie równań różniczkowych cząstkowych, obliczenia chemiczne, obliczenia finansowe a nawet obliczenia symboliczne. W wersji 6.0 zmieniono też procedury obliczeniowe na pochodzące z największej (zresztą darmowej) biblioteki procedur z dziedziny algebry liniowej LAPACK, a dyskretne przekształcenie Fouriera realizowane jest przez bibliotekę FFTW (także dostępną nieodpłatnie) – w rezultacie MATLAB staje się głównie językiem programowania i środowiskiem obliczeniowym, a w zakresie algorytmów numerycznych korzysta z najnowszych, sprawdzonych rozwiązań używanych szeroko w środowisku naukowym. MATLAB jest przy tym dostępny na platformy od Windows, Linux (niestety – jak na wszystkie inne platformy – komercyjnie) poprzez wszelkie warianty UNIX'a jak HP–UX, IRIX, AIX, Solaris i.t.p. aż po duże maszyny (mainframe) jak Cray (rozwój wersji do Macintosha wstrzymano na numerze 5.2).

Istnieją dość liczne programy naśladujące MATLAB'a, należące do kategorii freeware, na ogół nieco różniące się szczegółami składni[1] oraz pozbawione większości funkcji graficznych. Przykłady to Scilab, Octave, RLab, Euler.

[1] W konsekwencji nie mogą korzystać z toolboxów — nawet tych darmowych, chociaż np. do Octave jest wiele toolboxów oraz narzędzia pozwalające dokonać konwersji niektórych (tych darmowych) pakietów funkcji MATLAB'a

Rozdział 2

Filozofia korzystania z MATLAB'a

MATLAB, przynajmniej pierwotnie, przeznaczony był do pracy interaktywnej nad zadaniami numerycznej algebry liniowej. Z tego względu podstawowym elementem obsługi programu jest konsola tekstowa zwana zwykle oknem poleceń (ang. „command window"). Praca z programem polega na wypisywaniu poleceń zakończonych znakiem powrotu karetki („Enter"), które są niezwłocznie interpretowane przez program, a wyniki obliczeń wyświetlane w tym samym oknie. Polecenia mogą dotyczyć także otwarcia nowych okien do prezentacji graficznej wyników, a nawet okien wyposażonych w znany z Windows graficzny interfejs użytkownika, jednak konwersacja w trybie tekstowym pozostaje podstawową formą obliczeń. Najprostszym wariantem takiej konwersacji jest użycie MATLAB'a jako potężnego kalkulatora – obok znaku zachęty `>>` wpisujemy wyrażenie, kończąc je znakiem nowej linii („Enter"):

```
>> 2+sin(5*pi/7)-exp(3)

ans =

-17.3037

>>
```

Sposób prezentacji wyników można zmienić poleceniem `format`. Na przykład następująca sekwencja likwiduje dodatkowe puste linie w odpowiedzi programu, oraz zwiększa ilość wyświetlanych cyfr ułamkowych:

```
>> format compact
```

```
>> format long
>> 2+sin(5*pi/7)-exp(3)
ans =
 -17.30370544071964
```

Wyniki można zapamiętać w zmiennej o nazwie złożonej z maksymalnie 32 liter i cyfr (do wersji 4.2 włącznie do 20), zaczynającej się od litery

```
>> dluganazwa10=2+sin(5*pi/7)-exp(3);y=5,dluganazwa10+y^2
y=
   5
ans =
   7.69629455928036
```

Na uwagę zasługuje umieszczenie w jednej linii kilku instrukcji, oddzielonych przecinkiem lub średnikiem, oraz to, że rezultat instrukcji zakończonej średnikiem, nie jest kopiowany na wyjście.

W obliczeniach można używać liczb zespolonych:

```
>> (2+4*i)/(0.5-j)
ans =
 -2.40000000000000 + 3.20000000000000i
```

Większość funkcji standardowych rozszerzono tak, aby przyjmowały także argumenty zespolone.

```
>> sin(2+3*i)+exp(0.5-pi*j/3)/atan(2+i)
ans =
  9.66492370449750 - 5.45596908174169i
```

Dla przedstawienia jednostki urojonej zdefiniowano dwie funkcje `i` oraz `j` zwracające $\sqrt{-1}$. Można ich używać zamiennie, tak jak w powyższym przykładzie.

UWAGA! Istnienie funkcji o pewnej nazwie nie uniemożliwia utworzenia zmiennej o tej samej nazwie. Stwarza to pewne zagrożenie, gdyż litery „*i*" oraz „*j*" są zwyczajowo używane jako identyfikatory zmiennych przyjmujących wartości całkowite występujące przy realizacji pętli. Należy mieć to na uwadze prowadząc obliczenia w dziedzinie liczb zespolonych i unikać użycia liter „*i*" oraz „*j*" do organizacji pętli. W razie potrzeby można usunąć zmienne poleceniem `clear i j`.

Zdefiniowano funkcje `real`, `imag`, `conj`, `angle` zwracające odpowiednio część rzeczywistą i część urojoną liczby zespolonej, liczbę zespoloną sprzężoną do danej oraz argument liczby zespolonej. Moduł liczby zespolonej oblicza funkcja `abs`.

```
>> z=3+2*i
>> real(z),imag(z),conj(z),angle(z),abs(z)
ans =
     3
ans =
     2
ans =
  3.00000000000000 - 2.00000000000000i
ans =
   0.58800260354757
ans =
   3.60555127546399
```

Szczegółów o składni wywołania każdej funkcji można dowiedzieć się pisząc w linii poleceń `help <nazwa funkcji>`. Jeśli nie pamiętamy dokładnie nazwy funkcji można próbować wyszukać ją poleceniem `lookfor <hasło>`, co spowoduje wyszukanie wszystkich funkcji. w których opisie występuje „hasło". Na przykład w wersji 6.0 otrzymamy

```
>> help exp

 EXP    Exponential.
    EXP(X) is the exponential of the elements of X, e to the X.
    For complex Z=X+i*Y, EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y)).

    See also LOG, LOG10, EXPM, EXPINT.

>> lookfor exponent
EXP    Exponential.
EXPINT Exponential integral function.
EXPM   Matrix exponential.
EXPM1  Matrix exponential via Pade approximation.
EXPM2  Matrix exponential via Taylor series.
EXPM3  Matrix exponential via eigenvalues and eigenvectors.
BLKEXP Defines a function that returns the exponential of the input.
```

Rozdział 3

Zmienne: interpretacja macierzowa i tablicowa

Poniższy tekst nie obejmuje najnowszych konstrukcji języka i traktuje właściwie o wersji 4.2c, jednak ze względu na zgodność wstecz wszystkie omówione polecenia działają tak samo w wersjach o numerze większym od 5.0.

W starych wersjach MATLAB'a jedynym, a w najnowszych podstawowym typem danych jest tablica dwuwymiarowa liczb podwójnej precyzji (typu double). W istocie utworzone w przykładzie z poprzedniego rozdziału zmienne skalarne są zapamiętywane jako macierz 1×1. Zmienną o innych wymiarach tworzy się przy pomocy instrukcji przypisania i „konstruktora" macierzy w postaci nawiasów kwadratowych:

```
>> X=[1,2;3,4;5,6]
X=
   1 2
   3 4
   5 6
```

Powyższa instrukcja powoduje utworzenie zmiennej `X`, której wartością jest tablica o 3 wierszach i 2 kolumnach

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Jak widać tablicę (macierz) wprowadza się wierszami, które oddziela się średnikami, a elementy w wierszu oddzielane są przecinkami. Przecinki mogą zostać zastąpione spacjami (choć począwszy od wersji 6.0 zaleca się unikanie tej praktyki), a średniki znakiem nowej linii, tak więc następująca sekwencja poleceń wywołuje ten sam efekt

```
>> X=[1 2
>> 3 4
>> 5 6]
X=
   1 2
   3 4
   5 6
```

Elementy macierzy (tablicy) mogą być zadawane przy pomocy wyrażeń, mogą ponadto być liczbami zespolonymi.

```
>> X=[1, 2+3*i, 7+cos(pi/3); i, -3, exp(2)]
X =
   1.0000             2.0000 + 3.0000i   7.5000
        0 + 1.0000i  -3.0000             7.3891
```

Pewne specjalne macierze mogą być utworzone jako wynik funkcji standardowych. Do najważniejszych należą `eye`, `zeros`, `ones`, `rand`, `randn`. Funkcja `eye` służy do tworzenia macierzy jednostkowej wybranego wymiaru.

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

Kolejne dwie funkcje generują macierze wskazanych wymiarów złożone odpowiednio z samych zer i samych jedynek.

```
>> zeros(3,2),ones(2,3)
ans =
     0     0
     0     0
     0     0
ans =
     1     1     1
     1     1     1
```

Ostatnie dwie z wymienionych funkcji generują macierze pseudolosowe, przy czym dla funkcji `rand` elementy macierzy mają rozkład równomierny z przedziału [0, 1], natomiast dla funkcji `randn` – rozkład normalny o zerowej średniej i jednostkowej wariancji.

```
>> rand(3,2),randn(2,3)
ans =
   0.95012928514718   0.48598246870930
   0.23113851357429   0.89129896614890
   0.60684258354179   0.76209683302739
ans =
  -0.43256481152822   0.12533230647483  -1.14647135068146
  -1.66558437823810   0.28767642035855   1.19091546564300
```

Można oczywiście odczytać lub modyfikować wybrany element utworzonej zmiennej, stosując znaną z innych języków programowania notację (**UWAGA!** należy pamiętać, że nawiasy muszą być okrągłe!)

```
>> X(3,1)
ans =
   5
>> X(1,2)=7
X=
   1 7
   3 4
   5 6
```

Należy zwrócić uwagę, że pomimo zmodyfikowania tylko jednego elementu zmiennej, wyświetlana jest cała macierz `X`.

Możliwe jest także złączenie macierzy (tablic) w pionie lub poziomie, odpowiadające operacjom tworzenia macierzy blokowych:

```
>> X=[1,2;3,4],Y=[5,6,7;8,9,10],Z=[11,12;13,14;15,15]
X =
     1     2
     3     4
Y =
     5     6     7
     8     9    10
Z =
    11    12
    13    14
    15    15
>> A=[X,Y],B=[X;Z],C=[X,Y;Z,zeros(3,3)]
A =
     1     2     5     6     7
     3     4     8     9    10
```

```
B =
     1     2
     3     4
    11    12
    13    14
    15    15
C =
     1     2     5     6     7
     3     4     8     9    10
    11    12     0     0     0
    13    14     0     0     0
    15    15     0     0     0
```

Wymiary (liczbę wierszy i kolumn) zmiennej można odczytać przy pomocy funkcji `size`. Wywołana z jednym argumentem podaje ona dwuelementowy wektor wierszowy, którego pierwsza współrzędna jest liczbą wierszy, a druga liczbą kolumn[1]. Funkcję `size` można także wywołać z dwoma argumentami – drugi argument wskazuje wtedy, czy chcemy odczytać liczbę wierszy czy kolumn.

```
>> A=[1,2;3,4;5,6]
A =
     1     2
     3     4
     5     6
>> size(A)
ans =
     3     2
>> size(A,1)
ans =
     3
>> size(A,2)
ans =
     2
```

Najważniejszy operator jednoargumentowy działający na macierzach to „'" – hermitowskie sprzężenie macierzy, oznaczane zwykle w tekstach matematycznych gwiazdką w położeniu górnego indeksu ($\mathbf{A}^\star$) albo tak samo umieszczoną literą „H" ($\mathbf{A}^\mathrm{H}$). Zwykłą transpozycję (bez zamiany elementów zespolonych na sprzężone) uzyskuje się operatorem „.'".

[1] W wersji 5.0 i nowszych MATLAB'a możliwe są tablice uogólnione o większej liczbie wymiarów – dla takich zwracany wektor wymiarów jest dłuższy

```
>> X=[1, 2+3*i, 7+i; i, -3, 2-3*i]
X =
   1.0000             2.0000 + 3.0000i   7.0000 + 1.0000i
        0 + 1.0000i  -3.0000             2.0000 - 3.0000i
>> X'
ans =
   1.0000                  0 - 1.0000i
   2.0000 - 3.0000i  -3.0000
   7.0000 - 1.0000i   2.0000 + 3.0000i
>> X.'
ans =
   1.0000                  0 + 1.0000i
   2.0000 + 3.0000i  -3.0000
   7.0000 + 1.0000i   2.0000 - 3.0000i
```

Podstawowe operatory arytmetyczne „+", „−" wymagają jako argumentów tablic czy macierzy tych samych wymiarów i zwracają zgodnie z intuicją tablicę o tych samych wymiarach, co argumenty i elementach będących sumą czy różnicą odpowiednich elementów argumentów. W przypadku operatora mnożenia „∗" taka interpretacja „poelementowa" (będziemy dalej pisać – tablicowa) jest oczywiście możliwa, ale, ponieważ zasadniczym zastosowaniem MATLAB'a są obliczenia z dziedziny algebry liniowej, przyjęto, że gwiazdka oznacza mnożenie macierzowe: pierwszy czynnik musi mieć liczbę kolumn równą liczbie wierszy drugiego, a wynik ma tyle wierszy co pierwszy czynnik i tyle kolumn, co drugi, chyba że jeden z czynników jest skalarem – wtedy operacja jest zwykłym mnożeniem wektora lub macierzy przez skalar.

```
>> [1,2,3;4,5,6]+[1,4,9;16,25,36]
ans =
   2   6  12
  20  30  42
>> [1,2,4;8,16,32]*[4;2;1]
ans =
   12
   96
```

Powyższa sekwencja odpowiada następującym operacjom:

$$\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} + \begin{bmatrix}1 & 4 & 9\\ 16 & 25 & 36\end{bmatrix} = \begin{bmatrix}2 & 6 & 12\\ 20 & 30 & 42\end{bmatrix}$$

$$\begin{bmatrix}1 & 2 & 4\\ 8 & 16 & 32\end{bmatrix} \begin{bmatrix}4\\ 2\\ 1\end{bmatrix} = \begin{bmatrix}1\cdot 4 + 2\cdot 2 + 4\cdot 1\\ 8\cdot 4 + 16\cdot 2 + 32\cdot 1\end{bmatrix} = \begin{bmatrix}12\\ 96\end{bmatrix}$$

Obok operatora mnożenia traktującego argumenty jak wektory i macierze wprowadzono w MATLAB'ie także operator mnożenia tablicowego „.∗" wykonujący mnożenie „poelementowe" (operacja taka bywa niekiedy nazywana iloczynem Schura dwóch macierzy). Oba czynniki muszą mieć te same wymiary, a wynik jest tablicą tego samego wymiaru co każdy z czynników, złożoną z iloczynów odpowiadających sobie elementów.

```
>> [1,2,3;4,5,6].*[1,4,9;16,25,36]
ans =
   1     8    27
   64   125   216
```

Podobnie jak operator mnożenia potraktowano operator potęgowania „∧", który zastrzeżony jest dla przypadku, gdy jeden z argumentów jest skalarem, a drugi macierzą kwadratową. Do potęgowania tablicowego („poelementowego") służy operator „.∧"

I tak polecenie

```
>> [2,1;1,2]^2
ans =
   5 4
   4 5
```

odpowiada

$$\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}^2 = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

natomiast

```
>> 2^[2,1;1,2]
ans =
     5     3
     3     5
```

odpowiada

$$2^{\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}} = 2^{\left(\begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 \\ 0 & 3 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \right)} =$$

$$= \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \cdot \begin{bmatrix} 2^1 & 0 \\ 0 & 2^3 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$$

```
>> [2,1;1,2].^[2,3;4,5]
ans =
   4  1
   1 32

>> [1,2,3;4,5,6].^[6,5,4;3,2,1]
ans =
   1  32  81
  64  25   6
```

Od wspomnianych wyżej reguł dotyczących zgodności wymiarów argumentów jest jeden wyjątek: wtedy, gdy jeden z argumentów jest skalarem, a powinien być tablicą, tworzona jest niejawnie tablica odpowiedniego wymiaru o wszystkich elementach jednakowych, równych temu skalarowi. Zatem

```
>> [2,3,1;3,4,7].^2
ans =
   4   9   1
   9  16  49
```

daje taki sam wynik jak

```
>> [2,3,1;3,4,7].^[2,2,2;2,2,2]
ans =
   4   9   1
   9  16  49
```

MATLAB pozwala używać łańcuchów tekstowych. W starszych wersjach reprezentacja tekstu różniła się tylko jedną flagą, a każda litera była przechowywana jako liczba typu double, w nowszej (5.3) zastosowano UNICODE i reprezentację liter w postaci słów 16–bitowych. Zmienne tekstowe tworzy się przypisując im łańcuch ujęty w apostrofy

```
>> S='Tekst'
S=
   Tekst
```

Przetwarzanie łańcucha na wektor liczbowy osiąga się poleceniem `abs`

```
>> abs('ABCD')
ans =
   65   66  67   68
```

Możliwe jest też przekształcenie odwrotne poleceniem `asc`

```
>> asc([65,66,67,68])
ans =
   ABCD
```

W większości wypadków konwersje te były w starszych wersjach MATLAB'a istotne tylko przy wyprowadzaniu wyników na ekran, ale w najnowszych wersjach przestrzeganie właściwego typu danych jest konieczne dla funkcjonowania programu.

Możliwa jest konkatenacja (złączenie) łańcuchów w ten sam sposób co wektorów liczbowych.

```
>> ['test', 'TEST']
ans =
   testTEST
```

Nie będziemy omawiać szerzej możliwości przetwarzania danych tekstowych w MATLAB'ie, gdyż w prostych i najbardziej typowych zastosowaniach łańcuchy tekstowe potrzebne są jedynie do opisywania rysunków oraz wypisywania komunikatów na ekranie.

Rozdział 4

Zaawansowane indeksowanie

Obok prostego dostępu do pojedynczych elementów tablicy/macierzy, wspomnianego w poprzednim rozdziale, możliwe jest tworzenie bardziej wyrafinowanych wyrażeń indeksowych odpowiadających posługiwaniu się macierzami blokowymi. Można wskazać, by operacja była wykonana na elementach macierzy (tablicy) leżących na przecięciu wybranych wierszy i kolumn. W tym celu używa się wektorów zamiast pojedynczych liczb do indeksowania.

```
>> A=[1,2,3,4;5,6,7,8;9,10,11,12]
A =
   1   2   3   4
   5   6   7   8
   9  10  11  12
>> A([1,3],[1,3,4])
ans =
   1   3   4
   9  11  12
```

Jeśli chcemy wziąć pod uwagę wszystkie wiersze/kolumny nie ma potrzeby ich wypisywania – stosuje się dwukropek

```
>> A([2,3],:)
ans =
   5   6   7   8
   9  10  11  12
>> A(:,[1,4])
ans =
   1   4
   5   8
   9  12
```

Elementy (wiersze/kolumny) występują w takiej kolejności, w jakiej zostały wymienione, a zatem

```
>> A(:,[4,1])
ans =
   4   1
   8   5
  12   9
>> A([3,2],:)
ans =
   9  10  11  12
   5   6   7   8
>> A([3,1],[4,2])
ans =
  12  10
   4   2
```

Częste użycie w indeksowaniu postępów arytmetycznych spowodowało, że stworzono specjalną instrukcję „:” (dwukropek) konstruującą wektory (wierszowe), których współrzędne tworzą postęp arytmetyczny. Działa ona także poza wyrażeniami indeksowymi.

```
>> 1:5
ans =
   1   2   3   4   5
>> 3:6
ans =
   3   4   5   6
>> 3:2:9
ans =
   3   5   7   9
>> 3:2:10
ans =
   3   5   7   9
>> 7:-1:4
ans =
   7   6   5   4
>> 9:-3:4
ans =
   9   6
```

Wracając do poprzedniego przykładu możemy więc pisać

```
>> A(2:3,3:4)
ans =
   7   8
  11  12
```

Takie wyrażenia mogą, podobnie jak wyrażenie oznaczające pojedynczy element macierzy, występować także po lewej stronie symbolu przypisania:

```
>> A([1,3],1:3)=[4,8,16;32,64,128]
A =
   4   8  16   4
   5   6   7   8
  32  64 128  12
```

Obok tych sposobów indeksowania jest jeszcze jedna możliwość stosowana niekiedy do skomplikowanych manipulacji elementami tablic

```
>> B=[1,2;3,4;5,6]
B =
   1   2
   3   4
   5   6
>> B(:)
ans =
   1
   3
   5
   2
   4
   6
```

Jak widać z elementów tablicy `B` utworzony został w ten sposób jeden wektor kolumnowy, w którym kolejno ustawione zostały kolumny tablicy.

Ogólniejszą wersją powyżej omówionej konwersji jest polecenie `reshape`, które pozwala „nadać nowe wymiary macierzy”, pod warunkiem, że ilość elementów pozostaje niezmieniona.

```
>> A=[1,2;3,4;5,6]
A =
   1   2
   3   4
   5   6
>> B=reshape(A,3,2)
```

```
B =
   1   5   4
   3   2   6
>> A(:)-B(:)
ans =
   0
   0
   0
   0
   0
   0
```

Jak widać z powyższego przykładu nie zmienia się reprezentacja danych, a jedynie liczba wierszy i kolumn macierzy.

Rozdział 5

Pętle i instrukcje warunkowe

MATLAB nie byłby kompletnym językiem programowania, gdyby pozbawiony był instrukcji warunkowych oraz możliwości wielokrotnego wykonywania sekwencji instrukcji (pętli).

Instrukcje warunkowe przypominają znane z innych języków programowania

```
if wyrażenie logiczne,
   ...
[elseif wyrażenie logiczne]
   ...
[else]
   ...
end
```

gdzie wyrażenie logiczne jest uznawane za prawdziwe, gdy choć jeden jego element jest niezerowy. W MATLAB'ie występuje też znana z innych języków programowania instrukcja pętli while

```
while wyrażenie logiczne,
   ...
end
```

Nieco specyficzna jest instrukcja pętli `for`, wykonująca sekwencję instrukcji zadaną liczbę razy. Najczęstsze jej użycie wygląda zwykle następująco

```
for k=1:n,
   ...
end
for l=2:2:2*n,
   ...
end
```

Należy jednak pamiętać, że `p:r:k` oznacza wektor, którego elementy tworzą postęp arytmetyczny o pierwszym elemencie `p`, różnicy `r` i ostatnim elemencie `p+nr` najbliższym niewiększym od `k` gdy `r>0` i najbliższym niemniejszym, gdy `r<0`. W rozważanym kontekście można użyć dowolnego wektora wierszowego, a nawet tablicy (macierzy):

```
>> for k=[1,3,7], k, end
k =
   1
k =
   3
k =
   7
```

Jeżeli wyrażenie po prawej stronie symbolu przypisania w instrukcji `for` jest tablicą, to w kolejnych przebiegach pętli zmiennej sterującej przypisywane będą kolejne kolumny tej tablicy:

```
>> for v=[2,3,5;7,9,11], v, end
v =
   2
   7
v =
   3
   9
v =
   5
  11
```

Rozdział 6

Funkcje i skrypty; funkcje „inline”

Dowolną sekwencję instrukcji MATLAB'a można zapisać w pliku tekstowym z rozszerzeniem „.m”. Powstaje w ten sposób tak zwany M–skrypt. Wpisanie w linii poleceń nazwy pliku (bez rozszerzenia) powoduje wczytanie pliku i wykonanie zawartych w nim instrukcji. Operują one zawsze na głównej przestrzeni roboczej MATLAB'a i ich efekt jest identyczny, jakby wprowadzono je ręcznie z linii poleceń.

Użycie M–skryptów bywa wygodne, ale w przypadku wielokrotnie używanych w różnych okolicznościach sekwencji instrukcji należy rozważyć nadanie jej postaci funkcji użytkownika, tak zwanej M–funkcji. Jest to także sekwencja instrukcji zapisana w pliku tekstowym z rozszerzeniem „.m”, którego pierwsze linie mają postać

```
function [x,y,z,...]=nazwa(a,b,c,...)
% skrócony tekst pomocy (wyświetlany poleceniem lookfor)
% tekst pomocy (pełny)
% ............
% ............
% ............
% tekst pomocy
```

Taki nagłówek (wystarczy pierwsza linia) powoduje odmienne potraktowanie pliku. Wystąpienie nazwy pliku po raz pierwszy powoduje jego wczytanie do pamięci, a w razie wykrycia nagłówka funkcji zamianę zawartości na tak zwany p–kod, który pozostaje w pamięci aż do jawnego usunięcia poleceniem `clear nazwa` albo `clear functions` (w rzeczywistości problem jest nieco bardziej skomplikowany, gdyż przy wywołaniu sprawdza się, czy od czasu ostatniej kompilacji nie zmienił się plik źródłowy; sprawdzenie takie

nie dotyczy plików w podkatalogach katalogu `toolbox`, których dane katalogowe są przechowywane w pamięci podręcznej dla przyspieszenia dostępu w razie potrzeby odwołania się do nich). Jeśli użyto nazwy funkcji, która została już przekształcona do p–kodu, sprawdzone zostaje, czy lista parametrów aktualnych jest zgodna z listą parametrów formalnych. Następnie parametry aktualne wiązane są z formalnymi i wykonywany jest p–kod.

W stosunku do M–skryptu różnica polega na tym, że M–funkcja ma własną przestrzeń roboczą, a zmienne z głównej przestrzeni roboczej są dostępne tylko pod warunkiem jawnego zadeklarowania jako globalnych zarówno w głównej przestrzeni, jak i wewnątrz funkcji. Poza tym wszystkie zmienne mają charakter lokalny i przekazywanie wyników może odbyć się tylko poprzez parametry wyjściowe.

Przykład (plik „fun.m")

```
function y=fun(x)
%FUN - oblicza e^(x^2)+2*x-3
%
% funkcja obliczająca wyrażenie e^(x^2)+2*x-3
% zwektoryzowana
y=exp(x.^2)+2*x-3;
```

Umieszczenie pliku o powyższej zawartości w katalogu bieżącym MATLAB'a lub jednym z katalogów wymienionych na ścieżce wyszukiwania `matlabpath` pozwala używać tej funkcji następująco

```
>> fun(5)
   ans = 7.2005e+010
>> fun([0,1;2,3;4,5])
ans =
  1.0e+010 *
    0.0000    0.0000
    0.0000    0.0000
    0.0009    7.2005
>> format long e
>> fun([0,1;2,3;4,5])
ans =
   -2.000000000000000e+000    1.718281828459045e+000
    5.559815003314424e+001    8.106083927575384e+003
    8.886115520507872e+006    7.200489934438588e+010
```

Na uwagę zasługuje użycie operatora „.∧", które powoduje, że M–funkcja przy wywołaniu z argumentem będącym wektorem lub macierzą nie bę-

dzie sygnalizować błędu, ale zadziała podobnie do funkcji standardowych MATLAB'a.

Przykład (użycie zmiennych globalnych) Plik „fung.m"

```
function y=fung(x)
% FUNG oblicza trójmian kwadratowy o współczynnikach globalnych
%
% funkcja obliczająca wyrażenie a*x^2+b*x+c, gdzie a, b, c
% są skalarnymi zmiennymi globalnymi
global a b c
y=a*x.^2+b*x+c;
```

Użycie takiej funkcji wymaga wcześniejszego zadeklarowania, że zmienne `a`, `b`, `c` mają być traktowane jako globalne:

```
>> global a b c
>> a=2;b=3;c=5;
>> fung(3)
ans=
    29
>> fung([-1;0;2;3])
ans=
     4
     5
    19
    29
```

W wersji 5.3 MATLAB'a i nowszych dostępny jest nowy typ danych, który ułatwia definiowanie funkcji użytkownika: „inline". Łańcuch tekstowy zawierający proste wyrażenie może zostać zapamiętany jako obiekt, dla którego przeciążono operator indeksowania tak, że może być używany jak funkcja.

```
>> f=inline('x.*exp(-x.^2).*sin(x)','x')
f =
     Inline function:
     f(x) = x.*exp(-x.*2).*sin(x)
>> f(1),f([2,-1,-3])
ans =
   0.30955987565311
ans =
   0.03330872662439   0.30955987565311   0.00005224677764
```

Możliwe jest zautomatyzowanie „wektoryzacji” funkcji, to jest zastępowania operatorów mnożenia, dzielenia i potęgowania ich „tablicowymi” odpowiednikami:

```
>> g=inline('x*exp(-x^2)*sin(x)','x')
g =
     Inline function:
     g(x) = x*exp(-x^2)*sin(x)
>> gv=vectorize(g)
gv =
     Inline function:
     gv(x) = x.*exp(-x.^2).*sin(x)
>> g([-1;2])
??? Error using ==> inline/subsref
Error in inline expression ==> x*exp(-x^2)*sin(x)
??? Error using ==> ^
Matrix must be square.
>> gv([-1;2])
ans =
   0.30955987565311
   0.03330872662439
```

Rozdział 7

Algebra liniowa

Już pierwsza wersja MATLAB'a („classic Matlab") obok rozwiązywania układów równań liniowych udostępniała szereg zaawansowanych operacji na wektorach i macierzach oferowanych przez biblioteki LINPACK i EISPACK. W najnowszej wersji przepisane w języku C procedury z tych bibliotek zastąpiono biblioteką LAPACK – najobszerniejszym niekomercyjnym (http://www.netlib.org/lapack/) zestawem procedur z dziedziny algebry liniowej. Stąd obok operatorów arytmetycznych, „/" oraz „\" dostępnych jest szereg procedur, z których najważniejsze to `lu`, `chol`, `inv`, `rcond`, `cond`, `pinv`, `qr`, `eig`, `schur`, `svd`, `expm`. Ponadto polecenie `gallery` daje dostęp do szerokiego repertuaru funkcji generujących różnorodne specjalne macierze dogodne do testowania algorytmów tworzonych przez użytkownika.

7.1 Rozwiązywanie układów równań

Podstawowym narzędziem do rozwiązywania układów równań liniowych jest w MATLAB'ie operator „\".

```
>> A=[2,1;1,2],B=[3;-1]
>> A  B
ans =
   2.33333333333333
  -1.66666666666667
```

Rozwiązuje on równanie

$$\mathbf{Ax} = \mathbf{B}$$

gdzie $\mathbf{A}$ jest macierzą kwadratową (powiedzmy $n \times n$), zaś $\mathbf{B}$ macierzą o n wierszach (na przykład $n \times m$). Wynik jest macierzą o wymiarach $n \times m$, tak

jakby obliczono $\mathbf{A}^{-1}\mathbf{B}$, chociaż w rzeczywistości stosowana jest zazwyczaj metoda eliminacji Gaussa z częściowym wyborem elementów głównych w kolumnie. W pewnych przypadkach szczególnych stosowane są inne algorytmy: jeśli na przykład macierz $\mathbf{A}$ jest symetryczna, to dokonuje się sprawdzenia, czy jest dodatnio określona, a jeśli test wypadnie pomyślnie stosowany jest rozkład Choleskiego. Specjalne algorytmy stosowane są, gdy macierz $\mathbf{A}$ jest rzadka. Operator działa też wtedy, gdy macierz $\mathbf{A}$ jest prostokątna (powiedzmy o wymiarach $n \times k$), a w konsekwencji układ równań jest niedookreślony ($k > n$), albo nadokreślony ($k < n$). Wtedy stosowany jest rozkład QR z zamianą kolejności wierszy. Należy zwrócić uwagę, że jeśli $\mathbf{A}$ ma niepełny rząd, to `A\B` nie jest tym samym co pomnożenie $\mathbf{B}$ przez macierz pseudoodwrotną do $\mathbf{A}$.

Operator „/” działa podobnie – konstrukcja `B / A` jest równoważna `(A' \B')'`, a w większości wypadków odpowiada obliczeniu $\mathbf{B}\mathbf{A}^{-1}$.

7.2 Macierze odwrotne i pseudoodwrotne

Jakkolwiek rzadko w praktyce obliczeniowej naprawdę potrzebne jest jawne obliczenie macierzy odwrotnej, to oczywiście w MATLAB'ie przewidziano funkcję realizującą tę operację. Nazywa się ona `inv` i przyjmuje jeden argument, który powinien być macierzą kwadratową.

```
>> A=[2,-1;-1,2];B=inv(A)
B =
   0.66666666666667   0.33333333333333
   0.33333333333333   0.66666666666667
>> B*A,A*B
ans =
     1     0
     0     1
ans =
     1     0
     0     1
```

Ponadto dostępna jest funkcja `pinv` obliczająca macierz pseudoodwrotną Moore'a-Penrose'a.

```
>> A=[1,2,3;3,2,1]
>> B=pinv(A)
B =
  -0.16666666666667   0.33333333333333
```

```
   0.08333333333333   0.08333333333333
   0.33333333333333  -0.16666666666667
>> B*A,A*B
ans =
   0.83333333333333   0.33333333333333  -0.16666666666667
   0.33333333333333   0.33333333333333   0.33333333333333
  -0.16666666666667   0.33333333333333   0.83333333333333
ans =
   1.00000000000000  -0.00000000000000
   0.00000000000000   1.00000000000000
```

7.3 Rozkłady macierzy

Standardowo dostępne są funkcje obliczające rozkład LU macierzy kwadratowej, rozkład Choleskiego macierzy symetrycznej dodatnio określonej, rozkład QR dowolnej macierzy (także prostokątnej) oraz dekompozycję Schura macierzy kwadratowej i rozkład SVD dowolnej macierzy.

Rozkład trójkątny wykonywany jest, przez funkcję `lu`, w wersji z częściowym wyborem elementów głównych (w kolumnie), zatem dla danej macierzy $\mathbf{A}$ obliczane są: macierz permutacji $\mathbf{P}$, macierz trójkątna dolna $\mathbf{L}$ i macierz trójkątna górna $\mathbf{U}$ takie, że $\mathbf{A} = \mathbf{P}^{\mathrm{T}}\mathbf{L}\mathbf{U}$

```
>> A=[3,2,1;2,3,2;4,2,6]
A =
     3     2     1
     2     3     2
     4     2     6
>> [L,U,P]=lu(A)
L =
   1.00000000000000                   0                   0
   0.50000000000000   1.00000000000000                   0
   0.75000000000000   0.25000000000000   1.00000000000000
U =
   4.00000000000000   2.00000000000000   6.00000000000000
                   0   2.00000000000000  -1.00000000000000
                   0                   0  -3.25000000000000
P =
     0     0     1
     0     1     0
     1     0     0
>> P'*L*U
```

```
ans =
     3     2     1
     2     3     2
     4     2     6
```

Funkcja `qr` oblicza rozkład QR, czyli przedstawienie macierzy w postaci iloczynu macierzy ortogonalnej (unitarnej w przypadku zespolonym) i trójkątnej górnej:

```
>> [Q,R]=qr(A)
Q =
  -0.55708601453116   0.03851141912525  -0.82956135578434
  -0.37139067635410  -0.90501834944330   0.20739033894609
  -0.74278135270821   0.42362561037771   0.51847584736521
R =
  -5.38516480713450  -3.71390676354104  -5.75655548348861
                   0  -1.79078098932397   0.77022838250493
                   0                   0   2.69607440629911
>> Q'*Q
ans =
   1.00000000000000  -0.00000000000000  -0.00000000000000
  -0.00000000000000   1.00000000000000  -0.00000000000000
  -0.00000000000000  -0.00000000000000   1.00000000000000
>> Q*R
ans =
   3.00000000000000   2.00000000000000   1.00000000000000
   2.00000000000000   3.00000000000000   2.00000000000000
   4.00000000000000   2.00000000000000   6.00000000000000
```

Funkcja `schur` oblicza dekompozycję Schura, to znaczy znajduje macierz trójkątną górną unitarnie podobną do danej macierzy zespolonej $\mathbf{A}$, to jest oblicza takie macierze: unitarną $\mathbf{Q}$ i trójkątnej górnej $\mathbf{R}$, że $\mathbf{A} = \mathbf{QRQ}^\star$ (jeśli macierz wejściowa jest rzeczywista, to obliczana jest tak zwana rzeczywista postać Schura, w której $\mathbf{Q}$ jest ortogonalna, a $\mathbf{R}$ blokowo trójkątna górna z blokami na diagonali o wymiarach co najwyżej 2×2)

```
>> [Q,R]=schur(A)
Q =
  -0.77385266241444   0.63336565811076                   0
   0.28324973322991   0.34607743154558  -0.89442719099992
   0.56649946645982   0.69215486309116   0.44721359549996
R =
```

```
   1.53589838486225   2.68328157299975   1.03823229463674
                  0   8.46410161513775  -0.84974919968973
                  0                  0   2.00000000000000
>> Q'*Q
ans =
   1.00000000000000  -0.00000000000000   0.00000000000000
  -0.00000000000000   1.00000000000000   0.00000000000000
   0.00000000000000   0.00000000000000   1.00000000000000
>> Q*R*Q'
ans =
   3.00000000000000   2.00000000000000   1.00000000000000
   2.00000000000000   3.00000000000000   2.00000000000000
   4.00000000000000   2.00000000000000   6.00000000000000
```

Funkcja `svd` wyznacza rozkład SVD, czyli tak zwaną dekompozycję względem wartości szczególnych, to znaczy przedstawienie danej macierzy $\mathbf{A}$ w postaci iloczynu dwóch macierzy ortogonalnych (unitarnych w przypadku zespolonym) i macierzy, która ma niezerowe, a przy tym nieujemne, elementy tylko na głównej przekątnej ($\mathbf{A} = \mathbf{UDV}^\star$, gdzie $\mathbf{U}$, $\mathbf{V}$ są unitarne, $\mathbf{D}$ ma niezerowe elementy tylko na głównej przekątnej)

```
>> [U,D,V]=svd(A)
U =
  -0.36951928635138   0.63417547920808   0.67917373225528
  -0.42729451138351   0.53307779993731  -0.73023794735382
  -0.82515143916489  -0.56004421324141   0.07399717331702
D =
   8.90594290297009                  0                  0
                  0   2.51790456231010                  0
                  0                  0   1.15945574479352
V =
  -0.59103821974036   0.28932994341328   0.75296879527017
  -0.41222193148788   0.69402786662494  -0.59025282680259
  -0.69335914359483  -0.65925223108649  -0.29092884662910
>> svd(A)
ans =
   8.90594290297009
   2.51790456231010
   1.15945574479352
>> U'*U
ans =
   1.00000000000000   0.00000000000000   0.00000000000000
```

```
   0.00000000000000   1.00000000000000   0.00000000000000
   0.00000000000000   0.00000000000000   1.00000000000000
>> V'*V
ans =
   1.00000000000000   0.00000000000000   0.00000000000000
   0.00000000000000   1.00000000000000   0.00000000000000
   0.00000000000000   0.00000000000000   1.00000000000000
>> U*D*V'
ans =
   3.00000000000000   2.00000000000000   1.00000000000000
   2.00000000000000   3.00000000000000   2.00000000000000
   4.00000000000000   2.00000000000000   6.00000000000000
```

Funkcja `chol` działa tylko dla macierzy symetrycznych (hermitowskich w przypadku zespolonym) dodatnio określonych. Oblicza ona, dla danej macierzy $\mathbf{S}$ taką macierz trójkątną górną $\mathbf{R}$, że zachodzi $\mathbf{S} = \mathbf{R}^{\star}\mathbf{R}$

```
>> S=[3,2,1;2,3,2;1,2,3]
>> R=chol(S)
R =
   1.73205080756888   1.15470053837925   0.57735026918963
                  0   1.29099444873581   1.03279555898864
                  0                  0   1.26491106406735
>> R'*R
ans =
   3.00000000000000   2.00000000000000   1.00000000000000
   2.00000000000000   3.00000000000000   2.00000000000000
   1.00000000000000   2.00000000000000   3.00000000000000
```

7.4 Obliczanie wartości własnych

Do obliczania wartości własnych macierzy kwadratowych oraz uogólnionych wartości własnych pary macierzy kwadratowych przewidziano funkcję `eig`, wykorzystującą algorytm QR lub QZ

```
>> A=[1,2,3;-2,-4,5;6,7,0]
A =
     1     2     3
    -2    -4     5
     6     7     0
>> eig(A)
```

```
ans =
  -8.48331477354789
  -1.00000000000000
   6.48331477354788
```

Wywołana z dwoma argumentami wyjściowymi funkcja ta zwraca macierz prawych wektorów własnych oraz macierz diagonalną z wartościami własnymi na przekątnej.

```
>> [V,E]=eig(A)
V =
   0.04864853642211   0.76244373620987   0.53704590384728
   0.75324060269591  -0.64205788312410   0.27748368444451
  -0.65594352985169  -0.08025723539051   0.79660812325004
E =
  -8.48331477354789                   0                   0
                  0  -1.00000000000000                   0
                  0                   0   6.48331477354788
>> inv(V)*A*V
ans =
  -8.48331477354788  -0.00000000000000   0.00000000000000
   0.00000000000000  -1.00000000000000   0.00000000000000
  -0.00000000000000   0.00000000000000   6.48331477354788
```

UWAGA! Zaprezentowano jedynie najprostsze warianty korzystania z wymienionych funkcji. Większość standardowych funkcji MATLAB'a dopuszcza wywołanie ze zmienną liczbą argumentów, zarówno wejściowych jak wyjściowych, niekiedy dostarczając przy tym wyników silnie zależnych od sposobu wywołania, dlatego należy uważnie wpisywać odpowiednie polecenia, a najlepiej dodatkowo sprawdzać składnię wywołań poleceniem `help`.

7.5 Funkcje pomocnicze

Spośród setek najróżniejszych funkcji pomocniczych wymienimy tylko kilka: `norm` oblicza normy wektorów i macierzy, `cond` oblicza spektralną liczbę uwarunkowania macierzy kwadratowej (odwołując się do rozkładu SVD), zaś `rcond` szacuje odwrotność liczby uwarunkowania macierzy w normie $\|\cdot\|_1$.

Rozdział 8

Wielomiany w MATLAB'ie

W swej podstawowej wersji MATLAB oferuje kilka funkcji ułatwiających obliczenia z użyciem wielomianów (ponadto do najnowszych wersji istnieje Polynomial Matrix Toolbox wykorzystujący obiektowe rozszerzenia języka). Korzysta się z reprezentacji wielomianu w postaci wierszowego wektora współczynników uporządkowanych według malejących potęg zmiennej niezależnej, zatem

```
>> p=[3,2,-1,4,-5]
p=
  3   2  -1   4  -5
```

definiuje wektor, który użyty jako argument funkcji `roots`, `polyval`, `polyder`, `conv`, `deconv` i.t.p. reprezentuje wielomian $p(x) = 3x^4 + 2x^3 - x^2 + 4x - 5$. Funkcja `(polyval(p,x)` oblicza wartość wielomianu o współczynnikach zawartych w wektorze `p` w punktach wyspecyfikowanych w zmiennej `x` (funkcja jest zwektoryzowana względem drugiego argumentu) Przykład

```
>> polyval(p,-2)
ans =
      15
>> polyval(p,[-2,0,1,2])
ans =
      15  -5   3  63
```

Funkcja `roots` pozwala obliczyć pierwiastki wielomianu o współczynnikach umieszczonych w wektorze `p`, stosowana jest przy tym dość nietypowa metoda: budowana jest macierz kwadratowa (Frobeniusa) o wielomianie charakterystycznym równym danemu[1] i znajdowane są jej wartości własne (funkcja `eig` realizująca algorytm QR)

[1] podzielonemu przez współczynnik przy najwyższej potędze

```
>> roots(p)
ans =
  -1.6839
   0.1012 + 1.0975i
   0.1012 - 1.0975i
   0.8148
```

Funkcja `polyder(p)` oblicza wektor współczynników wielomianu będącego pochodną wielomianu reprezentowanego przez `p`.

```
>> polyder(p)
ans =
   12   6  -2   4
```

Funkcja `poly` generuje wektor współczynników wielomianu monicznego (to znaczy o współczynniku przy najwyższej potędze równym 1) i zadanych miejscach zerowych (pierwiastkach):

```
>> poly([1,2,3])
ans =
   1  -6  11  -6
>> poly([i,-i])
ans =
   1   0   1
>> roots(poly([-2,-1,1,2,3]))
ans =
    3.0000
    2.0000
   -2.0000
    1.0000
   -1.0000
```

Funkcja `conv` pozwala obliczyć wektor współczynników wielomianu będącego iloczynem wielomianów reprezentowanych przez jej argumenty

```
>> conv([1,1],[1,2,1])
ans =
     1     3     3     1
>> conv([1,2,1],[1,3,3,1])
ans =
     1     5    10    10     5     1
```

Funkcja `deconv` pozwala obliczyć iloraz i resztę z dzielenia jednego wielomianu przez drugi

```
>> deconv([1,5,10,10,6,2],[1,3,3,1])
ans =
     1     2     1
>> [q,r]=deconv([1,5,10,10,6,2],[1,3,3,1])
q =
     1     2     1
r =
     0     0     0     0     1     1
```

Rozdział 9

Przegląd niektórych funkcji standardowych MATLAB'a

Poniżej przedstawiono krótko wybrane standardowo dostępne funkcje MATLAB'a, które są wykorzystywane w trakcie Laboratorium Metod Numerycznych.

9.1 Interpolacja i aproksymacja

Główną funkcją wykorzystywaną w aproksymacji wielomianowej jest `polyfit`, która oblicza współczynniki wielomianu zadanego stopnia aproksymującego dane wejściowe optymalnie w sensie minimum sumy kwadratów błędów. Ponadto jeśli liczba węzłów jest o jeden większa od stopnia wielomianu, obliczany jest wielomian interpolacyjny (Lagrange'a).

```
>> p1=polyfit([0,1,2,3,4,5],[4.0000,3.3679,5.1353,9.0498,15.0183,23.0067],3)
p1 =
   -0.0202    1.2210   -1.7993    3.9918
>> p2=polyfit([0,1,2,3,4,5],[4.0000,3.3679,5.1353,9.0498,15.0183,23.0067],5)
p2 =
   -0.0008    0.0150   -0.1111    1.4408   -1.9759    4.0000
>> polyval(p1,[0,1,2,3,4,5])
ans =
    3.9918    3.3933    5.1157    9.0382   15.0397   22.9993
>> polyval(p2,[0,1,2,3,4,5])
ans =
    4.0000    3.3679    5.1353    9.0498   15.0183   23.0067
```

Należy pamiętać, że implementacja funkcji `polyfit` opiera się o konstrukcję macierzy Vandermonde'a i rozwiązanie układu równań z jej udziałem

(zwykle nadokreślonego – wyjątkiem jest sytuacja gdy zadany stopień wielomianu jest o jeden mniejszy od liczby węzłów, czyli mamy do czynienia z zadaniem interpolacji). Trzeba sobie z tego zdawać sprawę rozwiązując problemy aproksymacji z dużą liczbą równoodległych węzłów i wielomianem aproksymacyjnym wysokiego (powyżej 10) stopnia, gdyż w tych okolicznościach otrzymany wynik może być obarczony znacznym błędem ze względu na bardzo złe uwarunkowanie macierzy Vandermonde'a.

Warto pamiętać o istnieniu dwóch innych funkcji służących do interpolacji. Obie mają tę samą składnię wywołania – przyjmują trzy parametry wejściowe: wektor węzłów (muszą być uporządkowane rosnąco), wektor wartości funkcji interpolowanej w węzłach oraz wektor wartości zmiennej niezależnej, dla których ma być obliczona funkcja interpolująca. Wynikiem jest wektor, którego kolejnymi współrzędnymi są wartości funkcji interpolującej w wyspecyfikowanych punktach. W przypadku funkcji `spline` jest to funkcja sklejana sześcienna, natomiast w przypadku funkcji `interp1` ciągła funkcja kawałkami liniowa (w literaturze anglosaskiej taką interpolację nazywa się *lookup table*).

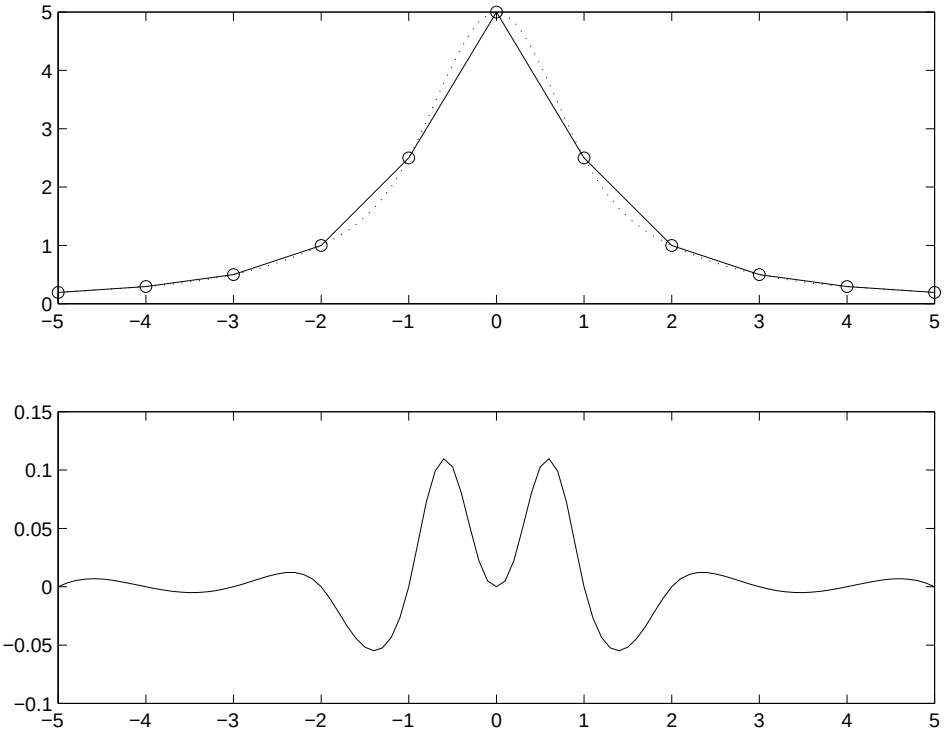
```
>> x=-5:5;y=5 ./((-5:5).^2+1);
>> xr=linspace(-5,5,101);yr=5./(x.*x+1);
>> ys=spline(x,y,xr);
>> subplot(2,1,1)
>> plot(x,y,'o',xr,yr,'-',xr,ys,':')
>> subplot(2,1,2)
>> plot(xr,ys-yr)
```

9.2 Całkowanie numeryczne

Repertuar funkcji standardowych obliczających przybliżone wartości całek oznaczonych funkcji jednej zmiennej ogranicza się do dwóch: `quad` oraz `quad8`. Mają one tę samą składnię wywołania: przyjmują jako parametry wejściowe nazwę M–funkcji obliczającej funkcję podcałkową, lewy i prawy koniec przedziału całkowania oraz tolerancję, z jaką ma być obliczony wynik. Obie procedury są adaptacyjne, a różnica między nimi polega na tym, że `quad` wykorzystuje metodę Simpsona, a `quad8` 8–punktowy wzór Newtona-Cotesa. Funkcja podcałkowa **musi** być zwektoryzowana.

Plik „fun.m"

```
function y=fun(x)
%
y=x.*sin(x);
```



Rysunek 9.1: Rysunek wygenerowany w powyższym przykładzie

```
>> S1=quad('fun',0,pi)
ans =
   3.11111000517089
>> S2=quad8('fun',0,pi)
ans =
   3.11109749786115
>> [S3,n]=quad('fun',0,pi,1e-4)
S3 =
   3.11109797851182
n =
    89
>> [S4,n]=quad8('fun',0,pi,1e-7)
S4 =
   3.11109749786115
n =
    33
```

Jak widać `quad8` może obliczyć tę samą całkę z większą dokładnością przy

mniejszej liczbie wywołań procedury obliczającej funkcję podcałkową, ale żadna z funkcji nie gwarantuje, że rzeczywisty błąd będzie mniejszy od założonej tolerancji.

W wersji 6.0 MATLAB'a omówione procedury zostały zmienione: adaptacyjna metoda Simpsona `quad` została istotnie udoskonalona w zakresie dokładności szacowania błędu oraz efektywności, natomiast funkcję `quad8` zastąpiono przez `quada` realizującą adaptacyjną metodę Lobbato. Składnia wywołania nie uległa większym zmianom. Autorem nowej wersji `quad` jest Walter Gander, a `quada` – Walter Gautschi. Jedynym zastrzeżeniem do autorów MATLAB'a może być brak procedury realizującej metody otwarte, które przydają się przy obliczaniu całek niewłaściwych (przez zamianę zmiennej) lub innych całek z funkcji nie dającej się obliczyć w końcu przedziału.

9.3 Rozwiązywanie równań różniczkowych zwyczajnych

W starszych wersjach MATLAB'a dostępne były dwie procedury do rozwiązywania zagadnień początkowych równań różniczkowych zwyczajnych: `ode23` oraz `ode45` realizujące metody Rungego-Kutty-Fehlberga z automatyczną zmianą długości kroku. Wymagały one doprowadzenia równania do postaci

$$\dot{y} = f(x, y) \qquad y(x_p) = y_0, \quad x \in [x_p, x_k] \tag{9.1}$$

Do wersji 4.2 Lawrence Shampine i Mark Reichelt opracowali pakiet procedur pod nazwą Matlab ODE Suite, znacznie rozszerzający możliwości MATLAB'a w omawianym zakresie. Począwszy od wersji 5.0 włączono ten pakiet do standardowej dystrybucji MATLAB'a. Zmieniła się składnia wywołania funkcji i wprowadzono szereg nowych procedur. Funkcja `ode23` wykorzystuje teraz „włożoną" (ang. *embedded*) parę metod typu Rungego-Kutty rzędu (2)3, autorstwa Shampine's-Bogackiego, natomiast `ode45` podobną parę rzędu (4)5 obliczoną przez Dormanda i Prince'a. Dodano ponadto funkcję `ode113`, będącą przeniesieniem do MATLAB'a (ale uwzględniającym jego specyfikę) znanych i cenionych procedur DE/STEP/INTERP napisanych w FORTRANie przez Shampine'a i Gordona. Procedura ta realizuje metody predyktor-korektor, bazujące na wzorach Adamsa z automatyczną zmianą kroku i rzędu metody. Drugim ważnym dodatkiem jest funkcja `ode15s` będąca implementacją tak zwanych metod numerycznego różniczkowania, mogąca też, przy odpowiednim wyborze opcji, realizować metody wstecznego różniczkowania (BDF) ze zmiennym krokiem i rzędem od 1 do 5. Obok `ode15s`, do rozwiązywania źle uwarunkowanych zagadnień początkowych (zwanych często

„sztywnymi"), służy `ode23s` implementująca „włożoną" parę Rungego-Kutty-Rosenbrocka, przeznaczoną do obliczeń z niewielką dokładnością. W późniejszych wersjach dodano jeszcze `ode23t` realizującą (niejawną) metodę trapezów i `ode23tb` opartą na połączeniu niejawnej metody trapezów z dwukrokową metodą wstecznego różniczkowania (TR-BDF2).

Wszystkie nowe procedury mają tę samą składnię wywołania oraz możliwości „gęstego" wyprowadzania wyników drogą interpolacji rozwiązania między obliczonymi punktami węzłowymi. W konsekwencji możemy zawsze zażądać podania wyników na dowolnie zdefiniowanej siatce wartości zmiennej niezależnej, nie wpływając na pracę algorytmu automatycznej zmiany długości kroku. W jednolity sposób przekazuje się do wszystkich funkcji opcje takie jak założona tolerancja obliczeń, sposób wyprowadzania wyników i.t.p. Niektóre dopuszczają ogólniejszą postać równania różniczkowego.

W najprostszej wersji rozwiązanie równania

$$\dot{y} = -xy \qquad y(0) = 10, \quad x \in [0,\, 5] \tag{9.2}$$

może być przeprowadzone następująco: należy przygotować M–funkcję obliczającą prawą stronę równania różniczkowego:

Należy utworzyć plik „fun.m":

```
function dy=fun(x,y)
%
dy=-x*y;
```

ewentualnie w wersji 5.3 można go zastąpić obiektem typu „inline"

```
>> f=inline('-x*y','x','y');
```

Następnie wywołać jedną z omówionych wyżej funkcji, na przykład `ode45`:

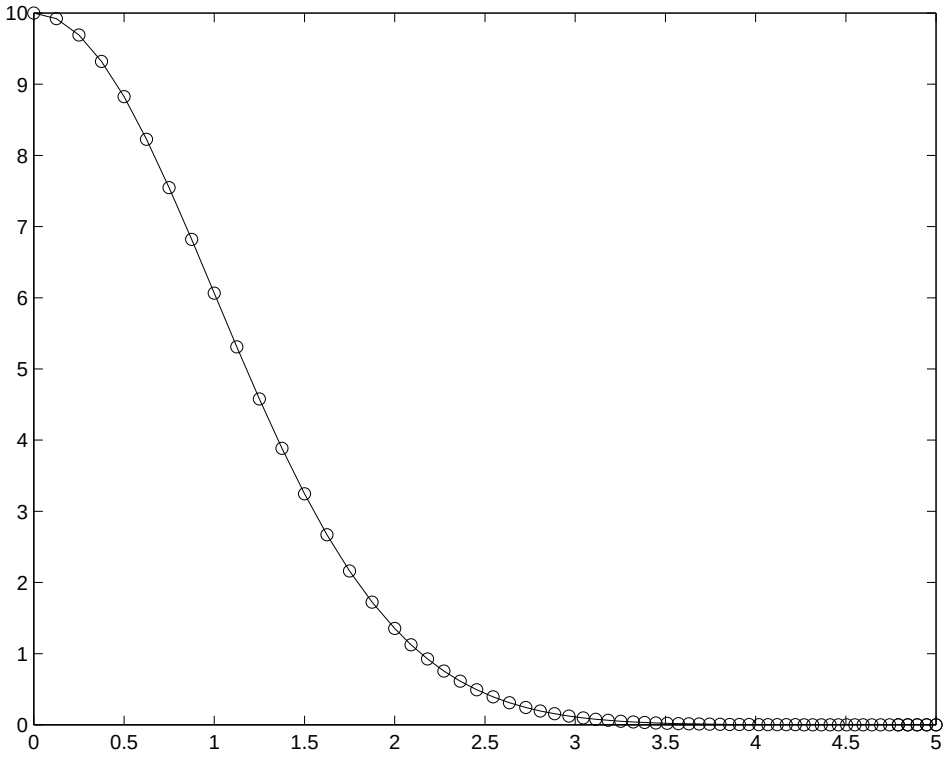
```
>> [x,y]=ode45('fun',[0,5],10);
```

(w przypadku obiektu „inline" apostrofy są zbędne: `[x,y]=ode45(f,[0,5],10);`)

Wywołane bez parametrów wyjściowych procedury rozwiązywania zagadnień początkowych wyświetlają rozwiązanie w formie graficznej:

```
>> ode45('fun',[0,5],10);
```

spowoduje otwarcie okna graficznego i narysowanie przebiegu widocznego na rysunku 9.2



Rysunek 9.2: Rozwiązanie równania $\dot{y} = -xy$ w przedziale $x \in [0, 5]$ z warunkiem początkowym $y(0) = 10$

Rozdział 10

Wybrane funkcje graficzne

Podstawową funkcją, służącą do graficznej prezentacji wyników jest `plot` i jej odmiany `loglog` – sporządzająca wykresy w skali logarytmicznej, oraz `semilogx` i `semilogy` – sporządzające wykresy w skali półlogarytmicznej. Podstawowa składnia wywołania to `plot(x,y)`, gdzie wektory `x` i `y` zawierają odpowiednio odcięte i rzędne wierzchołków łamanej, która ma zostać wykreślona. Na przekład

```
>> x=linspace(0,4*pi,101);y=sin(x);
>> plot(x,y)
```

kreśli dwa okresy sinusoidy.

Wywołanie funkcji `plot` można uzupełnić o trzeci parametr: łańcuch tekstowy określający kolor i rodzaj linii, jaka zostanie użyta do kreślenia. Kolory zakodowane są przy pomocy liter: „r" – czerwony, „g" – zielony, „b" – niebieski, „y" – żółty, „c" – jasnoniebieski (cyan), „m" – purpurowy (magenta), „w" – biały, „k" – czarny. Rodzaje linii wybiera się za pomocą kodów: „–" – ciągła, „– –"–przerywana, „:" – wielopunktowa (kropkowa), „.-"–punktowa. Zamiast linii można użyć markerów takich jak „+"–krzyżyk, „*" – gwiazdka, „x" – krzyżyk, „o" – kółeczko, „."–wypełnione kółeczko. W nowszych wersjach MATLAB'a (od 5.0 wzwyż) repertuar markerów jest większy i można łączyć je z liniami. Na przykład poniższa sekwencja poleceń `plot` wykreśli kolejno żółtą sinusoidę linią punktową, czerwoną kosinusoidę z krzyżykami w obliczonych punktach, zieloną sinusoidę linią punktową z kółkami w obliczonych punktach oraz purpurowe krzyżyki rozmieszczone na kosinusoidzie.

```
>> x=linspace(0,4*pi,101);y1=sin(x);y2=cos(x);
>> plot(x,y1,'y:')
>> plot(x,y2,'rx-')
>> plot(x,2*y1,'go:')
>> plot(x,2*y2,'m+')
```

W powyższym przykładzie nowy wykres zastępował poprzedni. W miarę potrzeby można dla każdego z wykresów otworzyć nowe okno graficzne poleceniem `figure` lub nałożyć na siebie umieszczając wszystkie, za wyjątkiem pierwszej, instrukcje rysowania między poleceniami `hold on` oraz `hold off`.

```
>> x=linspace(0,4*pi,101);y1=sin(x);y2=cos(x);
>> plot(x,y1,'y:')
>> hold on
>> plot(x,y2,'rx-')
>> plot(x,2*y1,'go:')
>> plot(x,2*y2,'m+')
>> hold off
```

Ten sam efekt można osiągnąć łącząc listy parametrów instrukcji `plot`:

```
>> x=linspace(0,4*pi,101);y1=sin(x);y2=cos(x);
>> plot(x,y1,'y:',x,y2,'rx-',x,2*y1,'go:',x,2*y2,'m+')
```

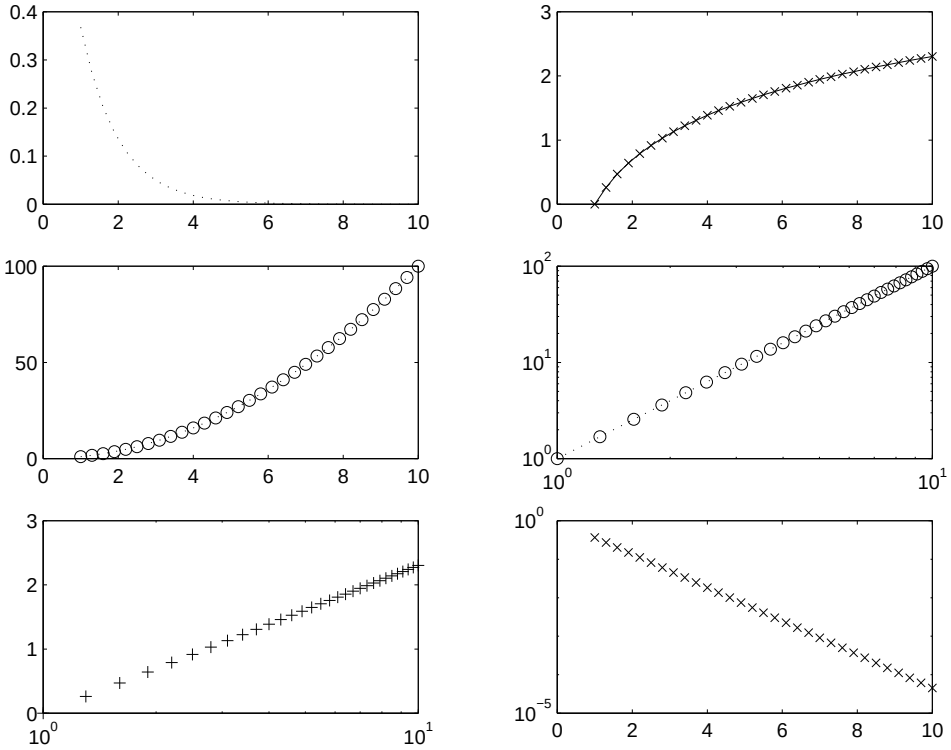
Możliwe jest także podzielenie okna graficznego na obszary, w których zostaną narysowane wykresy:

```
>> x=linspace(1,10,31);y1=exp(-x);y2=log(x);
>> subplot(3,2,1)
>> plot(x,y1,'b:')
>> subplot(3,2,2)
>> plot(x,y2,'rx-')
>> subplot(3,2,3)
>> plot(x,x.^2,'go:')
>> subplot(3,2,4)
>> loglog(x,x.^2,'go:')
>> subplot(3,2,5)
>> semilogx(x,y2,'m+')
>> subplot(3,2,6)
>> semilogy(x,y1,'cx')
```

Zawartość okna graficznego można wydrukować lub zapisać w postaci pliku w jednym z wielu formatów. Najbardziej użyteczne są Windows (Enhanced) Metafile, Postscript, Encapsulated Postscript, mapa bitowa (BMP), ewentualnie skompresowana mapa bitowa (GIF, JPEG).

```
>> print -dmeta -f1 nazwa.emf
>> print -deps -f2 nazwa.eps
>> print -dbitmap -f3 nazwa.bmp
```

zapisuje zawartość okien graficznych o numerach 1, 2 i 3 odpowiednio w postaci pliku Windows Metafile, Encapsulated PostScript oraz mapy bitowej w katalogu bieżącym. Numer okna graficznego jest zazwyczaj do odczytania w belce tytułowej, można go też zapamiętać (także w zmiennej) podczas tworzenia okna poleceniem `figure` (jest on wynikiem funkcji `figure`). Numer ostatnio użytego okna, lub okna wybranego kursorem myszki podaje ponadto funkcja `gcf` (w wersji 4.2b MATLAB'a można wprost dopisać „gcf" po „–f" w poleceniu `print`, jednak w nowszych wersjach zrezygnowano z tej możliwości – trzeba odczytać wynik funkcji `gcf`, a następnie wpisać do linii poleceń, lub używać innej składni wywołania polecenia `print` w połączeniu z funkcją `num2str`).



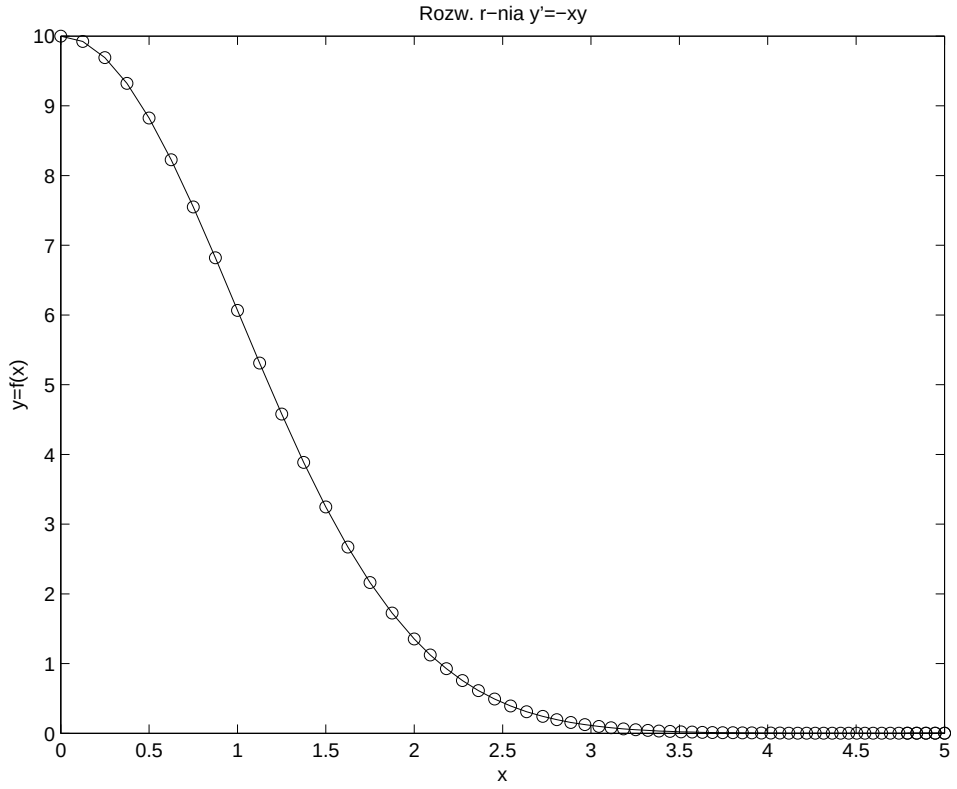
Rysunek 10.1: Rysunki wygenerowane w przykładzie ilustrującym działanie polecenia `subplot`

Dobrą praktyką jest opisywanie wykresów, które przeznaczone są do zapisania jako plik graficzny, co zapobiega później pomyłkom przy dołączaniu plików do dokumentu. Służyć mogą do tego celu polecenia `xlabel`, `ylabel` oraz `title`. Każde z nich, w swej podstawowej wersji, przyjmuje jako jedyny

argument łańcuch tekstowy i wyświetla napisy w bieżącym oknie graficznym. Na przykład sekwencja

```
>> xlabel('x')
>> ylabel('y=f(x)')
>> title('Rozw. r-nia y''=-xy')
```

powoduje, że rysunek 9.2 przybiera postać[1] 10.2.



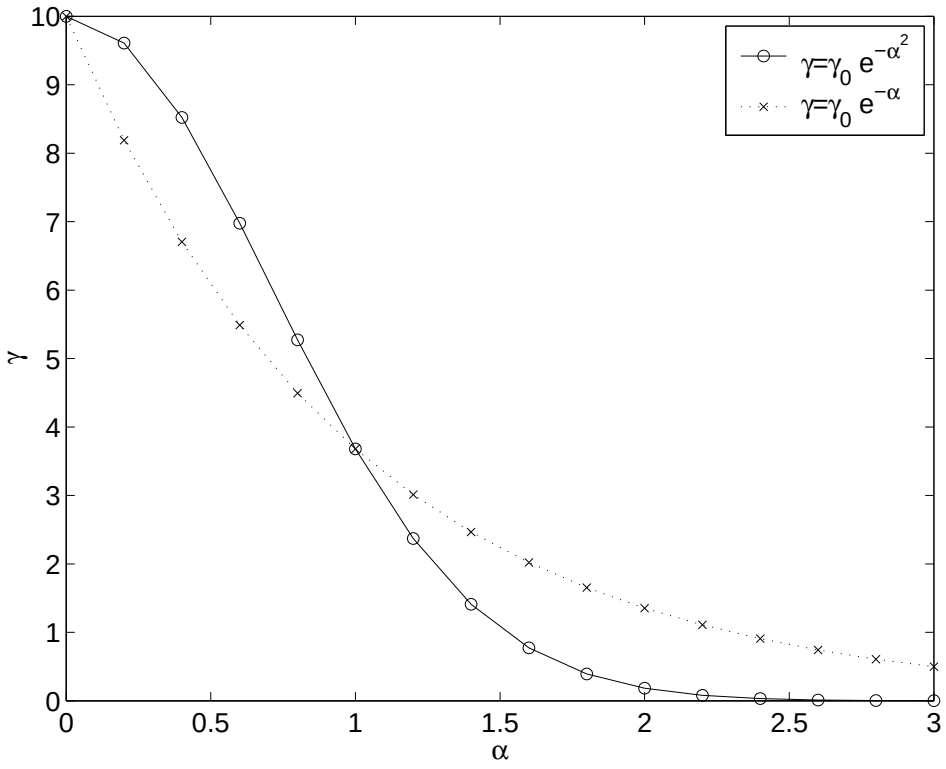
Rysunek 10.2: Rozwiązanie równania $\dot{y} = -xy$ opisane poleceniami `title`, `xlabel`, `ylabel`

Tych, którzy zetknęli się z pakietem LaTeX może zainteresować, że w opisach rysunków, począwszy od wersji 5.0 MATLAB'a, można użyć liter greckich oraz indeksów górnych i dolnych, w sposób znany z tego programu. Można więc na przykład użyć poleceń

[1] Niezadowolonym z domyślnej wielkości liter na rysunkach można polecić jej zmianę poleceniem `set(0,,'DefaultAxesFontSize',14)` **przed** przystąpieniem do generowania wykresów

```
>> x=linspace(0,3,16);
>> plot(x,10*exp(-x.^2),'-o',x,10*exp(-x),':x')
>> ylabel('\gamma')
>> xlabel('\alpha')
>> legend('\gamma=\gamma_0 e^{-\alpha^2}','\gamma=\gamma_0 e^{-\alpha}')
```

uzyskując rysunek 10.3



Rysunek 10.3: Prezentacja możliwości opisu rysunku poleceniem `legend`

Bibliografia

[1] Jerzy Brzózka, Lech Dorobczyński. *Programowanie w MATLAB.* Wydawnictwo MIKOM, 1998. ISBN 83-7158-120-3.

[2] Andrzej Zalewski, Rafał Cegieła. *MATLAB. Obliczenia numeryczne i ich zastosowania.* Biblioteka Użytkowników Mikrokomputerów. Wydawnictwo NAKOM, Poznań, 1996. ISBN 83-85060-85-5.

[3] Bogumiła Mrozek, Zbigniew Mrozek. *MATLAB. Uniwersalne środowisko do obliczeń naukowych i technicznych.* PLJ Warszawa, 1996. ISBN 83-7101-325-6.

[4] Bogumiła Mrozek, Zbigniew Mrozek. *MATLAB 5.X, SIMULINK 2.X. Poradnik użytkownika.* Wydawnictwo PLJ, 1998. ISBN 83-7101-376-0.

Spis treści