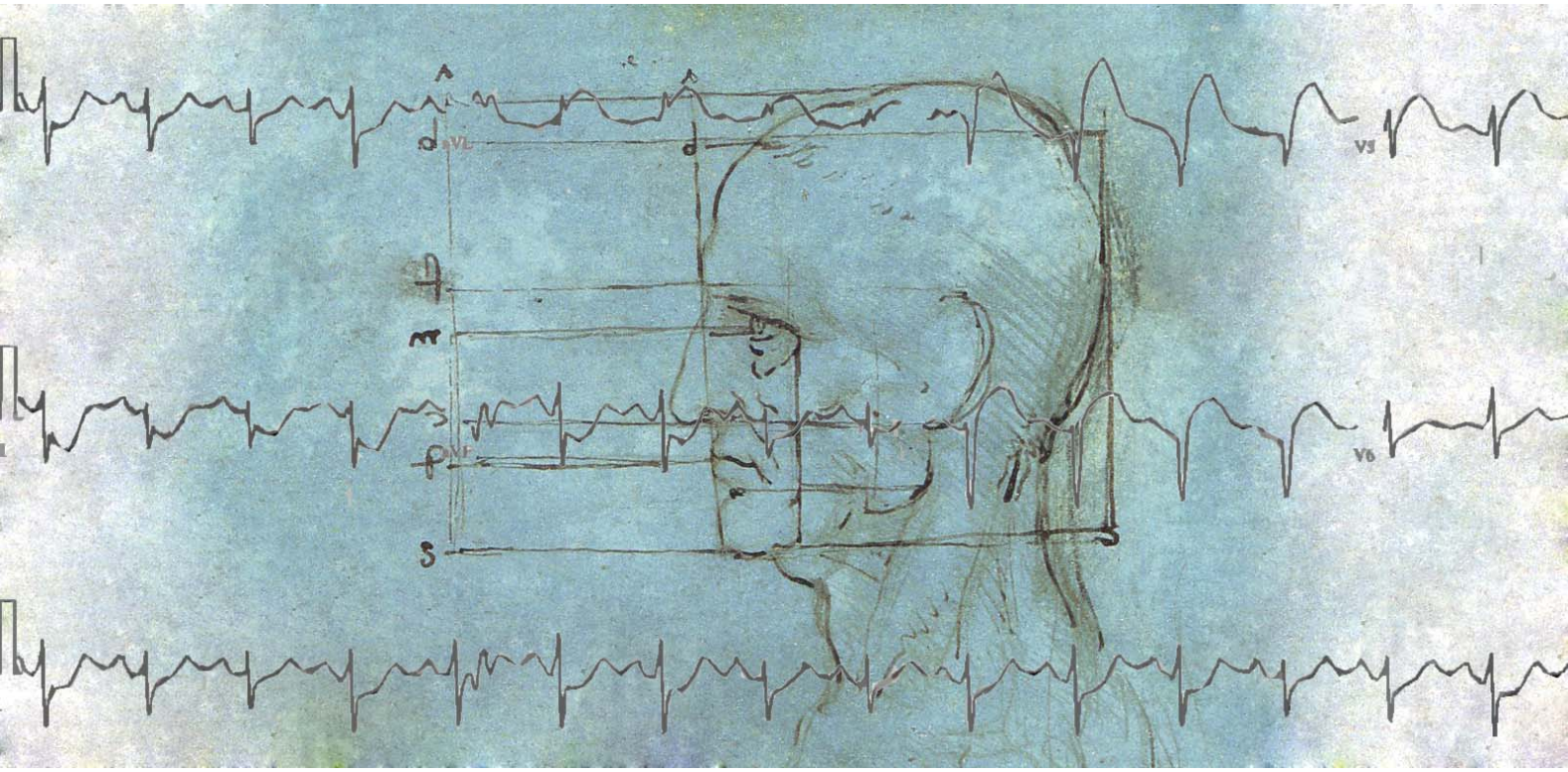

Metody obliczeniowe fizyki medycznej.

ćwiczenia, laboratorium

Wiesław Miklaszewski

Gdańsk, 2013

Wydział Matematyki, Fizyki i Informatyki
Uniwersytetu Gdańskiego

Skrypt powstał w ramach projektu „Kształcimy profesjonalistów – kompleksowy program kształcenia skierowany do studentów, absolwentów oraz studentów niepełnosprawnych Uniwersytetu Gdańskiego”
współfinansowanego przez Unię Europejską w ramach środków Europejskiego Funduszu Społecznego.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIWERSYTET GDAŃSKI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Metody obliczeniowe fizyki medycznej

Wiesław Miklaszewski

Rozdział 1

Podstawy MATLAB-a

1.1 Dlaczego MATLAB?

MATLAB, którego nazwa jest akronimem utworzonym z angielskiej nazwy Matrix Laboratory, jest językiem, programem komputerowym i interaktywnym środowiskiem przeznaczonym do przetwarzania danych w postaci tablic liczb. Środowisko MATLAB, dysponując licznym zestawem wbudowanych funkcji, ułatwia znakomicie prowadzenie obliczeń i wizualizację ich wyników.

1.2 Środowisko MATLAB

Po pierwszym[1] uruchomieniu programu MATLAB pojawiają się cztery okna (MATLAB R2011):

1. Górne prawe o angielskiej nazwie *Workspace* zawiera listę zmiennych zdefiowanych przez użytkownika i ich własności;

2. Centralne okno o nazwie *Command* służy do wprowadzania poleceń języka MATLAB;

3. Dolne prawe (*Command History*) zawiera wszystkie polecenia wydane przez użytkownika w oknie *Command* w czasie wszystkich sesji[2] użytkownika[3];

4. Lewe (*Current Folder*) zawiera listę plików z bieżącego foldera.

[1] Po następnych uruchomieniach MATLAB odtwarza zestaw okien z końca poprzedniej sesji. Okna mogą być zamykane lub otwierane poleceniami z menu *Desktop*

[2] Sesja to okres między włączeniem i wyłączeniem programu MATLAB

[3] Historia poleceń może zostać skasowana instrukcją *Clear Command History* z menu *Edit* (to samo dotyczy okien *Workspace* i *Command*)

MATLAB jest środowiskiem interaktywnym, tzn. użytkownik może wydawać mu kolejno polecenia (tak jak prostemu kalkulatorowi), ale bardziej sensowne jest tworzenie i uruchamianie programów (skryptów). Narzędziem, służącym do budowanie, testowania i uruchamiania skryptów jest *Editor/Debugger*, który możemy otworzyć klikając na iconę *New script*, umieszczoną pod menu *File*, korzystając ze skrótu *Ctrl+N*, używając polecenia *New* z menu *File*, itd.

Dostęp do innych okien umożliwia menu *Desktop*. Aby zamknąć dowolne okno, kliknij na symbol „×" w jego górnym prawym rogu. Optymalnym sposobem konfiguracji zestawu okien jest jednoczesne używanie okien *Command* i *Editor* oraz otwieranie innych okien w razie potrzeby.

1.3 Praca w oknie *Command*

W czasie każdej sesji MATLAB przechowuje w swojej pamięci aktualne wartości wszystkich zmiennych zdefiniwanych przez wszystkie wyrażenia wprowadzone w okienku *Command* lub obliczone w czasie uruchamianie skryptów. Zmienne te mogą być usunięte przez wywołanie funkcji *clear* lub wybór polecania *Clear Workspace* z menu *Edit*. Wartości przypisane zmiennym są dostępne w czasie sesji poprzez wywołanie nazwy zmiennej lub użycie tej nazwy w wyrażeniu.

Wszystkie wyrażenia wykonane w okienku *Command* zostają w tym oknie (aż do wyczerpania przeznaczonej na to pamięci) i mogą być przywołane przez przewijanie w tył. Nie są natomiast dostępne wyrażenia wykonane w czasie realizacji skryptu. Zawartość okna *Command* może być usunięta poleceniem *Clear Command Window* z menu *Edit* lub funkcją *clc*.

Funkcja *whos* wypisuje nazwy i atrybuty wszystkich aktualnie zdefinowanych zmiennych. Analogiczna informacja jest dostępna w oknie *Workspace*

Wywołanie większości funkcji MATLAB-a powoduje wypisanie w okienku *Command* jednej lub wielu liczb. Sposób ich reprezantacji może być regulowany opcjami funkcji *format*[4]. Na przykład polecenie

$$format\ compact$$

usuwa puste linie, które z założenia oddzielają komunikaty wyświetlane w okienku *Command*, natomiast sekwencja

$$format\ long\ e$$

powoduje zmianę standardowego sposobu wyświetlania liczb (5 cyfr) na format 16-cyfrowy z 3-cyfrowym wykładnikiem. Powrót do standardowego stylu listowania zapewnia polecenie

$$format\ short.$$

[4]To samo można osiągnąć używając polecenia *Preferences* z menu *File/Command Window*

Inne opcje funkcji *format* są przestawione w tabeli 1.1.

Tabela 1.1: Przykładowe wydruki po użyciu niektórych opcji funkcji *format*

Opcja	Liczba > 1	Liczba < 1
short	333.3333	0.0033
short e	3.3333e+10	3.3333e-07
short g	33.333	0.00033333
short eng	3.3333e+006	3.3333e-003
long	3.333333333333333e+07	0.003333333333333
long e	3.333333333333333e+07	3.333333333333300e-03
long g	3333333.3333333	0.0033333333333333333
long eng	3.33333333333330e+006	3.33333333333333e-003
rational	1000/3	1/300
hex	4074d555554c62af	3f6b4e81b4e81b4f
bank	333.33	0.00

Użycie skrótu *Ctrl+C* przerywa realizowany lub zatrzymany program.

1.4 Obliczanie wyrażeń w oknie *Commad* – elementarna składnia

MATLAB pozwala użytkownikom tworzyć zmienne o nazwach, których długość nie przekracza 63. znaków. Każda taka nazwa musi zaczynać sie od litery. Pozostałe znaki mogą być literami, cyframi lub znakiem „_”. MATLAB rozróżnia małe i wielkie litery.

Jednak istnieją nazwy, które są zarezerwowane dla języka programowanie MATLAB. Te nazwy, wyszczególnione w tabeli 1.2, nie mogą być używane jako nazwy zmiennych.

Tabela 1.2: Nazwy zarezerwowne dla języka progrmowania MATLAB

break	*global*	*case*	*if*	*catch*	*otherwise*
continue	*persistent*	*else*	*return*	*elseif*	*switch*
end	*try*	*for*	*while*	*function*	

Jeżeli używamy okienka *Command* do obliczenia wyrażenia, które zawiera zmienne, musimy je uprzednio zdefinować, przypisując im wartości. Operatorem przypisania jest znak równości „=”. Należy obok znaku zachęty programu MATLAB „»” wprowadzić nazwę zmiennej, znak „=”, wartość numeryczną i nacisnąć *Enter*, np. (wcześniej wykonano polecenie *format compact*)

```
»a = 8.2                    Użytkownik naciska Enter
a=
        8.2000              Odpowiedź systemu
```

Średnik (;) umieszczony na końcu wyrażenia wyłącza odpowiedź systemu

```
»a = 8.2;
»b = -9.3;
»c = 10.4;
»
```

MATLAB pozwala umieszczać kilka wyrażeń w jednej linii. W takim wypadku wyrażenia są oddzielone przecinkiem (,) lub średnikiem (;). Jeżeli zostanie użyty przecinek, to system wyświetla odpowiedź, jeżeli średnik, to odpowiedzi nie będzie.

1.4.1 Operatory arytmetyczne

Pięć operatorów arytmetycznych +, -, *, / i ^ pozwala na wykonywanie dodawania, odejmowania, mnożenia, dzielenia i potęgowania, odpowiednio. Na przykład wyrażenie matematyczne

$$d = \frac{a^2}{1 + bc}$$

może być zakodowane w MATLAB-ie jako

```
d=a^2/(1+b*c)
```

Oczywiście wielkości skalarne a, b i c muszą mieć przypisane wcześniej wartości numeryczne. Jeżeli tego nie uczyniono, pojawi się komunikat o błędzie.

1.4.2 Kolejność wykonywania operacji matematycznych

Stosowanie nawiasów zapewnia prawidłową hierarchię wykonywania operacji matematycznych. Jeżeli nie ma nawiasów, to w pierwszej kolejności jest wykonywane potęgowanie, potem mnożenie i dzielenie, a następnie dodawanie i odejmowanie. Wewnątrz nawiasów działania na każdym poziomie hierarchii są wykonywane od lewej do prawej.

1.4.3 Skalary a tablice

Ponieważ MATLAB traktuje zmienne jak tablice liczb, wszystkie operacje arytmetyczne muszą być zgodne regułami algebry liniowej. Kiedy zmienne są skalarami, tzn. są tablicami jednoelementowymi (jeden wiersz i jedna kolumna), to stosują się do nich reguły algebry. Jednak można działać na tablice

liczb, nie stosując reguł algebry liniowej, jeżeli użyje się operacji kropkowanych, które wykonują sekwencje operacji arytmetycznych na tablicach tego samego rozmiaru element jednej tablicy przez element drugiej. Kiedy używamy operatorów kropkowanych, mnożenie, dzielenie i potęgowanie mają postać: .*, ./ i .^.

1.4.4 Zmienna *ans*

Gdy okno *Command* jest używane jak kalkulator i obliczane wyrażenie nie jest przypisane żadnej zmiennej, MATLAB zawsze przypisuje odpowiedź zmiennej o nazwie *ans*. Ta zmienna może być teraz używana jak każda inna. Obliczenie następnego wyrażenia bez przypisania go zmiennej zmienia aktulaną wartość zmiennej *ans*.

1.4.5 Liczby zespolone

Liczby zespolone są reprezentowane w MATLAB-ie za pomocą symboli i i j, które oznaczą jednostkę urojoną $\sqrt{-1}$. Na przykład

$$2.0000 + 3.0000i$$

MATLAB pozwala na mieszanie liczb rzeczywistych i zespolonych, np.

$$z = 4 + sqrt(-4)$$

1.5 Funkcje MATLAB-a

MATLAB dostarcza bardzo wielu funkcji elementarnych i specjalnych. Niektóre z nich są opisane w tabelach 1.3 i 1.4, a predefiniowane stałe w tabeli 1.5.

Tabela 1.3: Niektóre funkcje elementarne MATLAB-a

Funkcja matematyczna	Reprezentacja w MATLAB-ie
e^x	*exp(x)*
$e^x - 1$	*expm1(x)*
$\sqrt{x}$	*sqrt(x)*
$\ln x$	*log(x)*
$\ln(1 + x)$	*log1p(x)*
$\log_{10} x$	*log10(x)*
$\|x\|$	*abs(x)*
$sgn(x)$	*sign(x)*
$n!$	*factorial(n)*
Liczby pierwsze $\leqslant n$	*primes(n)*

Tabela 1.4: Funkcje trygonometryczne i hiperboliczne MATLAB-a

	Trygonometryczne			Hiperboliczne	
Funkcja	x[rad]	x [°]	Odwrotna		Odwrotna
sinus	*sin(x)*	*sind(x)*	*asin(x)*	*sinh(x)*	*asinh(x)*
cosinus	*cos(x)*	*cosd(x)*	*acos(x)*	*cosh(x)*	*acosh(x)*
tangens	*tan(x)*	*tand(x)*	*atan(x)*	*tanh(x)*	*atanh(x)*
cotangens	*cot(x)*	*cotd(x)*	*acot(x)*	*coth(x)*	*acoth(x)*
secans	*sec(x)*	*secd(x)*	*asec(x)*	*sech(x)*	*asech(x)*
cosecans	*csc(x)*	*cscd(x)*	*acsc(x)*	*csch(x)*	*acsch(x)*

Tabela 1.5: Stałe MATLAB-a

Wielkość	Wyrażenie MATLAB-a	Komentarz
π	pi	3,141592653589793
$\sqrt{-1}$	i lub j	używana do konstukcji liczby zespolonej $a+ib$, gdzie a i b są rzeczywiste
Względna dokładność zmiennoprzecinkowa	*eps*	Odległość od 1,0 do następnej większej liczy zmiennoprzecinkowej ($\approx 2.22 \times 10^{-16}$)
∞	*inf*	
$0/0$, $0 \times \infty$ i $\infty \times \infty$	*NaN*	Wynik niezdefiniowanej operacji matematycznej
Największa liczba zmiennoprzecinkowa	*realmax*	$\approx 1.7977e + 308$
Najmniejsza dodatnia liczba zmiennoprzecinkowa	*realmin*	$\approx 2.2251e - 308$

Tabela 1.6: Relacje w MATLAB-bie

Relacja	Symbol	Symbol MATLAB-a
Równość	$=$	==
Nierówność	$\neq$	~=
Mniejszy	$<$	<
Mniejszy równy	$\leqslant$	<=
Większy	$>$	>
Większy równy	$\geqslant$	>=

W tabeli 1.6 zamieszczono operatory relacji używane w MATLAB-ie.

MATLAB umożliwia zaokrąglanie liczb w różny sposób. Funkcje wykonujące te operacje są opisane w tabeli 1.7.

Funkcje umożliwiające tworzenie i wykonywanie różnych operacji na licz-

Tabela 1.7: MATLAB – funkcje zaokrąglające

Funkcja	Argument (x)	Wynik (y)	Zaokrąglanie
$y = fix(x)$	3.8 -3.8 $3.81 - 3.79i$	3 -3 $3.0000 - 3.0000i$	w kierunku zera
$y = round(x)$	3.8 -3.8 $3.81 - 3.79i$	4 -4 $4.0000 - 4.0000i$	do najbliższej liczby całkowitej
$y = ceil(x)$	3.8 -3.8 $3.81 - 3.79i$	4 -3 $4.0000 - 3.0000i$	kierunku nieskończoności
$y = floor(x)$	3.8 -3.8 $3.81 - 3.79i$	3 -4 $3.0000 - 4.0000i$	w kierunku minus nieskończoności

bach zespolonych są opisane w tabeli 1.8.

Tabela 1.8: MATLAB – operacje na liczbach zespolonych

Funkcja	z	y	Opis
$z = complex(a, b)$	$a + bi$		Tworzenie liczby zespolonej, a i b rzeczywiste
$y = abs(z)$	$2 + 3i$	3.6056	Moduł liczby zespolonej $\sqrt{a^2 + b^2}$
$y = angle(z)$	$a + bi$	$atan2(b, a)$	Faza w radianach
$y = real(z))$	$2 + 3i$	2	Część rzeczywista
$y = imag(z)$	$2 + 3i$	3	Część urojona
$y = conj(z)$	$2 + 3i$	$2 - 3i$	Sprzężenie zespolone

W tabli 1.9 przedstawiono niektóre funkcje specjalne, a w tabeli 1.10 elementarne funkcje statystyczne.

Tabela 1.9: MATLAB – niektóre funkcje specjalne

Funkcja	Funkcja MATLAB-a	Opis
$Ai(x), Bi(x)$	$airy(0,x)$, $airy(2,x)$	Funkcje Airy'ego
$I_v(x)$	$besseli(v,x)$	Zmodyfikowana funkcja Bessela pierwszego rodzaju
$J_v(x)$	$besselj(v,x)$	Funkcja Bessela pierwszego rodzaju
$K_v(x)$	$besselk(v,x)$	Zmodyfikowana funkcja Bessela drugiego rodzaju
$Y_v(x)$	$bessely(v,x)$	Funkcja Bessela drugiego rodzaju
$B(x,w)$	$beta(x,w)$	Funkcja beta
$K(m)$, $E(m)$	$ellipke(m)$	Zupełne całki eliptyczne pierwszego i drugiego rodzaju
$erf(x)$, $erfc(x)$	$erf(x)$, $erfc(x)$	Funkcja błędu i uzupełniająca funkcja błędu
$E_1(z)$	$expint(x)$	Funkcja całkowo-wykładnicza
$\Gamma(x)$	$gamma(x)$	Funkcja gamma
$P_n^m(x)$	$legendre(n,x)$	Stowarzyszona funkcja Legendre'a

Oprócz pięciu operatorów arytmetycznych (+, -, *, / i ^) w MATLAB-ie stosuje się również inne symbole. Ich znacznie i sposób użycia jest opisany w tabeli 1.11.

Tabela 1.11: MATLAB – znaki specjalne

Znak	Nazwa	Sposób użycia
.	kropka	(1) Kropka dziesiętna (2) część operatora arytmetycznego wskazująca specjalny typ operacji tablicowej, np. a.*b (3) znak oddzielający w strukturze, np. *pacjent.nazwisko*
		kontynuacja na następnej stronie

Tabela 1.11: MATLAB – znaki specjalne (kontynuacja)

Znak	Nazwa	Sposób użycia
,	przecinek	(1) separator w elemencie macierzy, np. $a(3,6)$, w funkcji, np. $legendre(n,x)$, w nawiasach tworzących wektor, np. [3,6,9] lub w outpucie funkcji, np. $[C,I]=max(x)$ (2) umieszczany na końcu wyrażenia, kiedy kilka wyrażeń pojawia się w jednej linii
;	średnik	(1) umieszczony na końcu wrażenia wstrzymuje wyświetlanie jego wyniku (2) oznacza koniec wiersza w poleceniu tworzenia macierzy, np. $A=[158;345]$
:	dwukropek	(1) separator w poleceniu tworzenia wektora, np. $a=1:0.1:10$ (2) oznacza „wszystkie wiersze” – $A(:,j)$ lub „wszystkie kolumny” – $A(i,:)$ macierzy
()	nawiasy okrągłe	(1) zawierają indeksy elementu macierzy, np. $A(3,1)$ (2) zawierają elementy wyrażeń matematycznych, np. $(a+1)\hat{}3$ (3) zawierają argumenty funkcji, np. $sin(x)$
[ ]	nawiasy kwadratowe	tworzą tablicę liczb (wektor albo macierz) lub łańcuchów znaków
{ }	nawiasy klamrowe	(1) tworzą tablicę komórek, np. $aC=\{[1.7,6.5,5.6],rand(7)/3,'arka'\}$ (2) zawierają indeksy elementów tablicy komórek, np. $aC\{2\}$
%	procent	oznacza początek komentarza, tzn. MATLAB ignoruje wszystkie znaki po jego prawej stronie
%%	procenty	oznacza początek lub koniec komórki (fragmentu kodu) w *edytorze* MATLAB-a
%{ %}	procenty i nawiasy klamrowe	zawierają blok linii komentarza
’	apostrof	(1) $'arka'$ oznacza łańcuch znaków $arka$ (2) oznacza sprzężenie hermitowskie wektora lub macierzy
...	wielokropek	oznacza kontynuację wyrażenia w następnej linii
	spacja	w zależności od kontekstu jest ignorowana, oznacza separator w poleceniu tworzenia, np. [3 5 7], jest elementem łańcucha lub oddziela argumenty pewnych funkcji MATLABA-a, np. $format\ long$
		kontynuacja na następnej stronie

Tabela 1.11: MATLAB – znaki specjalne (kontynuacja)

Znak	Nazwa	Sposób użycia
@	„małpa”	tworzy uchwyt funkcji, np. @$nazwafunkcji$
\	ukośnik lewy	(1) macierzowy operator matematyczny (2) używany do generacji greckich liter i symboli matematycznych w grafice

1.6 Przeciążanie

MATLAB pozwala na przeciążanie nazw wbudowanych lub utworzonych przez użytkownika funkcji, np. następujące wyrażenie jest w MATLAB-ie dozwolone

$$sin = ax + b * y^3;$$

1.7 Tworzenie skryptów i ich wykonywanie – edytor MATLAB-a

Skrypt jest plikiem zawierającym listę poleceń, które będą wykonywane tak jakby były wywołane w oknie *Command*. Można go utworzyć w dowolnym procesorze lub edytorze tekstu albo w edytorze MATLAB-a i zapisać jako plik tekstowy z rozszrzeniem „.m” (stąd nazwa $m\text{-}file$). Jeżeli użyjemy procesora lub edytora tekstu, to skrypt można uruchomić pisząc jego nazwę bez rozszerzenia w oknie *Command*. To samo dotyczy edytora MATLAB-a, ale możemy również użyć menu tego edytora. Nazwy skryptów podlegają tym samym ograniczeniom co nazwy zmiennych. Edytor MATLAB-a automatycznie przypisuje nazwie skryptu rozszerzenie „.m”.

Innym rodzajem plików tworzonych w tym edytorze są pliki zawierające funkcje. Są one wymagane przez niektóre wbudowane funkcje MATLAB-a, ale przede wszystkim umożliwiają programowanie strukturalne. Różnią się one od wyrażeń wprowadzanych w linii komend tym, że MATLAB przydziela im prywatną przestrzeń roboczą i formalnie zdefiniowanymi relacjami wejścia-wyjścia w środowisku MATLAB-a.

Używamy skryptów, gdy program zawiera więcej niż kilka linii lub będzie wykonywany wielokrotnie.

1.8 Edytor MATLAB-a

Edytor MATLAB-a ma kilka własności, które powodują, że jest bardzo wygodnym narzędziem do tworzenia plików ze skryptami i funkcjami, a mia-

Tabela 1.10: MATLAB – niektóre funkcje statystyczne

Funkcja	Funkcja MATLAB-a	Opis
maksymalna wartość x	$max(x)$	Największy/e element/y tablicy
minimalna wartość x	$min(x)$	Najmniejszy/e element/y tablicy
$\langle x \rangle$	$mean(x)$	Średnia wartość tablicy wartości
mediana	$median(x)$	Mediana tablicy wartości
dominanta	$mode(x)$	Najczęściej występujące wartości w tablicy wartości
σ	$std(x)$	Odchylenie standardowe tablicy wartości
σ^2	$var(x)$	Wariancja tablicy wartości

nowicie:

- pozwala na łatwe konwertowanie kodu na komentarz i z powrotem (menu *Text*);
- dostarcza pomocy wizualnej, kiedy tworzymy struktury programu;
- pozwala na automatyczne wcinanie tekstu (menu *Text*);
- grupuje nawiasy;
- posiada system sprawdzający składnię w czasie rzeczywistym (*Code Analyzer*) – włącznie/wyłącznie – menu *Text/Preferences*;
- pozwala na definiowanie fragmentów kodu (komórek – *cells*), które mogą być wykonywane niezależnie od innych fragmentów; komórki są wydzielane dwoma znakami %% na jej początku i końcu (aktywcja/dezaktywacja – menu *Cell*).

1.9 Wykonywanie programów

MATLAB musi wiedzieć, gdzie znajdują się skrypty i funkcje, które ma wykonać. Tę informację możemy mu przekazać używając menu *File/Set Path*. Jeżeli spróbujemy wykonać skrypt, który nie znajduje się w bieżącym folderze, z edytora, MATLAB zapyta się czy chcemy dokonać zmiany tego foldera. Tej zmiany można również dokonać z okna *Command* używając interfejsu *Current Path*.

Programy mogą wyświetlać wyniki w okienku *Command* w różny sposób:

- możemy nie umieszczać średnika (;) na końcu wyrażenia;
- możemy użyć polecenia *disp*;
- możemy użyć polecenia $fprintf$.

Uruchamiając skryt warto „oczyścić otoczenie", tzn.

- usunąć wszystkie zmienne poleceniem *clear all*;
- usunąć zawartość okna *Command* poleceniem *clc*;
- zamknąć okienka graficzne poleceniem *close all*.

1.10 System pomocy

MATLAB posiada rozbudowany system pomocy, który można uruchomić na kilka sposobów:

- klikając na ikonę (?) w pasku okna *Command*;
- naciskając klawisz $F1$,
- używając menu $Help$;
- pisząć w okienku *Command* polecenie $help\ nazwafunkcji$.

1.11 Tworzenie wektorów

Wektory wierszowe w MATLAB-ie są zapisywane jako

$$x = [x_1\ x_2\ x_3\ \ldots\ x_n]$$

lub

$$x = [x_1, x_2, x_3, \ldots, x_n]$$

gdzie x_i są liczbami, nazwami zmiennych, wyrażeniami lub łańcuchami.

1.12 Operator (:)

MATLAB dostarcza kilku innych sposobów przypisywania wartości elementom wektora. Jednym z nich jest notacja dwukropkowa, pozwalająca na wyspecyfikowanie zakresu wartości i różnicy między kolejnymi elementami wektora, tzn.

$$x = x_0 : \Delta x : x_k$$

lub

$$x = (x_0 : \Delta x : x_k)$$

albo

$$x = [x_0 : \Delta x : x_k]$$

gdzie x_0 – wartość początkowa, Δx – różnica i x_k – wartość końcowa mogą być dowolnymi kombinacjami wartości liczbowych, nazw zmiennych i wyrażeń. W wyniku tej operacji zostanie utworzony wektor x postaci

$$[x_0, x_0 + \Delta x, x_0 + 2\Delta x, \ldots, x_0 + n\Delta x],$$

gdzie $x_0 + n\Delta x \leqslant x_k$. Jeżeli Δx nie jest wyspecyfikowany, MATLAB przyjmuje $\Delta x = 1$, a więc polecenie

$$x = x_1 : x_k$$

utworzy wektor

$$x = [x_0, x_0 + 1, x_0 + 2, \ldots, x_0 + n]$$

Liczbę elementów wektora można znaleźć używając funkcji MATLAB-a

$$length(x)$$

Możemy utworzyć wektor kolumnowy transponując wektor wierszowy, np.

$$x = (0 : 0.1 : 2)'$$

gdzie (') oznacza operację transpozycji.

1.13 Tworzenie wektora o n równoodległych wartościach

Następujące polecenie utworzy wektor o n równoodległych wartościach

$$x = linspace(x_0, x_k, n)$$

gdzie x_0 – pierwszy element wektora, a x_k – ostatni. Różnica między elementami jest obliczana przez MATLAB ze wzoru

$$\Delta x = \frac{x_k - x_0}{n - 1}.$$

A zatem *linspace* generuje wektor

$$x = [x_0, x_0 + \Delta x, x_0 + 2\Delta x, \ldots, x_0 + (n-1)\Delta x]$$

Oczywiście $x_k = x_0 + (n-1)\Delta x$ jest elementem wektora. Jeżeli n nie jest wyspecyfikowane, MATLAB przyjmuje, że $n = 100$.

Jeżeli potrzebny jest wektor o równoodległych elementach w skali logarytmicznej, mamy do dyspozycji polecenie

$$x = logspace(x_0, x_k, n)$$

które tworzy wektor wierszowy

$$x = [10^{x_0}, 10^{x_0+\Delta x}, 10^{x_0+2\Delta x}, \ldots 10^{x_k}]$$

Jeżeli n nie jest podane, MATLAB przyjmuje $n = 50$.

1.14 Operator (') dla wektorów o elementach zespolonych

Jeżeli wektor ma elementy zespolone, operator (') tworzy wektor sprzężony po hermitowsku, tzn. transponuje go i dokonuje sprzężenia zespolonego składowych, np. polecenie

$$z = [2, 9 + 3i, -26.7, 4.2 - 0.27i]'$$

da wektor kolumnowy

$z=$
2
9-3i
-26.7
4.2+0.27i

Jeżeli chcemy dokonać tylko transpozycji takiego wektora musimy dodatkowo sprząc go zespolenie poleceniem *conj*, tzn.

$$z = conj([2, 9 + 3i, -26.7, 4.2 - 0.27i]')$$

1.15 Dostęp do elementów wektora

Jeśli chcemy uzyskać dostęp do elementów wektora musimy użyć notacji indeksowej, która dla wektora $x = [x_1, x_2, x_3, \ldots, x_n]$ ma postać $x(i)$, co odpowiada składowej x_i wektora, $i = 1, 2, \ldots, n$. Aby uzyskać dostęp do ostatniego elementu wektora, można użyć słowa kluczowego *end*, np.

$$xostatni = x(end)$$

Na dostęp do grup elementów pozwala operator (:)

$$x(p : r : k)$$

gdzie $1 \leqslant p < k \leqslant n$, p – indeks początkowy i k – końcowy są dodatnimi liczbami całkowitymi; w tym wypadku również różnica indeksów r jest też taką liczbą. Na przykład jeżeli $n = 11$, to możemy wybrać elementy wektora x począwszy od drugiego do szóstego poleceniem

$$x(2 : 6)$$

Tutaj brak parametru r oznacza, że $r = 1$. Inną metodą jest użycie wektora indeksów, np. jeżeli chcemy uzyskać dostęp do elementów: 2-ego, 3-ego, 6-ego i 9-ego wektora x, możemy wykonać polecenie

$$y = x([2, 3, 6, 9])$$

1.16 Praca z wektorami

Jeżeli dodamy/odejmiemy skalar do/od wektora, to zostanie on dodany/odjęty do/od każdego elementu wektora, np. wykonanie

$$x = [-3, 4, 2, 5, 8, 9, 11] + 2$$

da w rezultacie

$x=$
-1 6 4 7 10 11 12

Aby pomożyć drugi i trzeci element wektora przez liczbę oraz wstawić wynik na miejsce trzeciego i czwartego elementu, odpowiednio, możemy użyć notacji dwukropkowej

$$x(3:4) = x(2:3) * 4$$

Sortujemy wektor używając polecenia *sort*

$$[xnowy, indeksy] = sort(x)$$

Otrzymamy posortowany od najmniejszego od największego elementu wektor $xnowy$ i odpowiadający wektor indeksów $indeksy$, zawierający pierwotne położenia posortowanych elementów. Polecenie

$$[xnowy, indeksy] = sort(x,' descend')$$

wygeneruje wektor posortowany w odwrotnym porządku.

Funkcja

$$find(relacja)$$

znajduje położenia (nie wartości) wszystkich elementów wektora, które spełniają warunek reprezentowany przez wyrażenie $relacja$, np.

$$indeksy = find([-2, 5, 16, -8, 32, 1, -7, -7] <= 0)$$

wygeneruje wektor

$indeksy=$
-1 4 7 8

Jedną z zalet MATLAB-a jest niejawna notacja wektorowa (macierzowa), pozwalajaca na kodowanie w sposób zwarty operacji na elementach wektora, np. jeżeli zdefiniujemy wektor

$$x = linspace(-2 * pi, 2 * pi, 25)$$

polecenie

$$y = sin(x)$$

da wektor zawierający sinusy elementów wektora x.

MATLAB dostarcza funkcji znajdujących ekstremalne elementy wektora. Aby znaleźć wartość i położenie najmniejszego/największego elementu wektora x, używamy odpowiednio konstrukcji

$$[xmin, ideksmin] = min(x)$$

$$[xmaks, ideksmaks] = max(x)$$

1.17 Tworzenie macierzy

Rozważmy następującą macierz

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix}.$$

Można ją utworzyć w MATLAB-ie na kilka sposobów, np.

$$A = [a_{11}, a_{12}, a_{13}; a_{21}, a_{22}, a_{23}; a_{31}, a_{32}, a_{33}; a_{41}, a_{42}, a_{43}]$$

gdzie średnik (;) oznacza koniec wiersza, lub w bardziej czytelnej formie (znak kontynuacji „...” może być zastąpiony naciśnięciem klawisza *Enter*)

$$\begin{aligned} A = [a_{11}, \quad & a_{12}, \quad a_{13}; \ldots \\ & a_{21}, \quad a_{22}, \quad a_{23}; \ldots \\ & a_{31}, \quad a_{32}, \quad a_{33}; \ldots \\ & a_{41} \quad a_{42} \quad a_{43}] \end{aligned}$$

Ponieważ możemy traktować macierz jako wektor kolumnowy jej wierszy, to następujący ciąg instukcji utworzy nam macierz

$$x_1 = [a_{11}, a_{12}, a_{13}]$$

$$x_2 = [a_{21}, a_{22}, a_{23}]$$

$$x_3 = [a_{31}, a_{32}, a_{33}]$$

$$x_4 = [a_{41}, a_{42}, a_{43}]$$

$$A = [x_1; x_2; x_3; x_4]$$

We wszystkich powyższych wyrażeniach a_{ij} oznaczają liczby, nazwy zmiennych, wyrażenia lub łańcuchy.

Liczbę wierszy m i kolumn n macierzy można uzyskać używając polecenia

$$[m, n] = size(A)$$

Zastosowanie funkcji *length* do macierzy da jej liczbę kolumn.

Sprzężnie hermitowskie macierzy jest realizowane za pomocą operatora (').

MATLAB dostarcza dwóch funkcji, *repmat* i *meshgrid*, które mogą być używane do tworzenia macierzy poprzez replikację skalara, wektora (kolumnowego albo wierszowego), macierzy lub łańcucha. Polecenie

$$repmat(x, r, c)$$

spowoduje utworzenie macierzy, składającej się z r kopii skalara, wektora lub macierzy x, umieszczonych w wierszu i c takich kopii wstawionych w kolumnie, np.

$$repmat(22.13, 1, 5)$$

da wektor

22.1300 22.1300 22.1300 22.1300 22.1300

a polecenie

$$repmat(22.13, 2, 2)$$

macierz (2×2)

$$\begin{matrix} 22.1300 & 22.1300 \\ 22.1300 & 22.1300 \end{matrix}$$

Pierwszy argument tej funkcji może być wektorem, wtedy np. polecenie

$$repmat([1, 2], 3, 2)$$

wygeneruje macierz (3×4)

$$\begin{matrix} 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 \\ 1 & 2 & 1 & 2 \end{matrix}$$

itd.

Jeżeli mamy dwa wektory $x = [x_1, x_2, x_3, x_4]$ i $y = [y_1, y_2, y_3]$, to używając polecenia

$$[A, B] = meshgrid(x, y)$$

otrzymamy dwie macierze

$$A = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \\ x_1 & x_2 & x_3 & x_4 \end{pmatrix}$$

i

$$B = \begin{pmatrix} y_1 & y_1 & y_1 & y_1 \\ y_2 & y_2 & y_2 & y_2 \\ y_3 & y_3 & y_3 & y_3 \end{pmatrix}$$

W obu przypadkach zostały utworzone macierze o wymiarze $(length(y) \times length(x))$.

1.18 Tworzenie macierzy specjalnych

- Macierz $(n \times m)$ o wszystkich elementach równych 1

  $$ones(n, m)$$

- Macierz (n, m) o wszystkich elementach równych 0

  $$zeros(n, m)$$

- Aby utworzyć macierz diagonalną, której elementami diagonalnymi są elementy wektora x, używamy polecnia

  $$diag(x)$$

  Ta funkcja służy również do otrzymania diagonalnych elementów macierzy A o rozmiarze $(n \times n)$, tzn. polecenie

  $$x = diag(A)$$

  da wektor o długości n zawierający diagonalne elementy macierzy A.

- Macierz identycznościowa

  $$eye(n)$$

  gdzie n oznacza jej rozmiar.

- Kwadrat magiczny – sumy wszystkich elementów w każdej kolumnie, w każdym wierszu i na diagonali są równe

  $$magic(n)$$

  gdzie n oznacza jej wymiar.

- Inne macierze specjalne – p. $Help$

  $$gallery$$

1.19 Dostęp do elementów macierzy

- Element macierzy
$$A(i, j)$$
- Wszystkie elementy j-tej kolumny
$$A(:, j)$$
- Wszystkie elementy i-tego wiersza
$$A(i, :)$$
- „Podmacierz": wiersze od i_p do i_k, kolumny od j_p do j_k
$$A(i_p : i_k, j_p : j_k)$$
- „Podmacierz": wiersze od i_p do i_k co i_r-ty, kolumny od j_p do j_k co j_r-ta
$$A(i_p : i_r : i_k, j_p : j_r : j_k)$$
- Tworzenie wektorów indeksów
$$[indeksy_{wiersze}, indeksy_{kolumny}] = find(relacja)$$
gdzie $indeksy_{wiersze}$ i $indeksy_{kolumny}$ to wektory kolumnowe określające położenie elementów spełniających warunek $relacja$

1.20 Ekstremalne elementy macierzy

Minimalne i maksymalne elementy macierzy uzyskujemy stosując polecenie min i max, odpowiednio. Funkcje te dają na wyjściu wektory, które zawierają najmniejsze (największe) elementy w każdej kolumnie macierzy.

1.21 Permutacje cykliczne wierszy i kolumn macierzy

Rozważmy macierz
$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}.$$
Polecenie
$$fliplr(A)$$

daje macierz o odwróconej kolejności kolumn, tzn.

$$\begin{pmatrix} a_{13} & a_{12} & a_{11} \\ a_{23} & a_{22} & a_{21} \\ a_{33} & a_{32} & a_{31} \end{pmatrix}.$$

natomiast komenda

$$flipud(A)$$

odwraca kolejność wierszy, czyli powstanie macierz

$$\begin{pmatrix} a_{31} & a_{32} & a_{33} \\ a_{21} & a_{22} & a_{23} \\ a_{11} & a_{12} & a_{13} \end{pmatrix}.$$

1.22 Operacje na macierzach

Rozważmy macierze

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Polecenie $A \pm B$ doda (odejmie) te macierze, dając

$$\begin{pmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} \end{pmatrix}.$$

Komenda $[A, B]$ dołączy kolumny macierzy B do macierzy A i otrzymamy

$$\begin{pmatrix} a_{11} & a_{12} & b_{11} & b_{12} \\ a_{21} & a_{22} & b_{21} & b_{22} \end{pmatrix},$$

natomiast wyrażenie $[A; B]$ dołączy wiersze macierzy B do macierzy A, tzn. powstanie macierz

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}.$$

Oczywiście macierz może zostać pomnożona/podzielona przez skalar, tzn. każdy jej element zostanie pomnożony/podzielony przez ten skalar

1.22.1 Operacje kropkowe

Operator (.) pozwala na przeprowadzenie na macierzach o tych samych rozmiarach działań arytmetycznych element przez element, a więc

$$A.*B = \begin{pmatrix} a_{11}*b_{11} & a_{12}*b_{12} \\ a_{21}*b_{21} & a_{22}*b_{22} \end{pmatrix},$$

$$A./B = \begin{pmatrix} a_{11}/b_{11} & a_{12}/b_{12} \\ a_{21}/b_{21} & a_{22}/b_{22} \end{pmatrix},$$

$$A.\hat{}B = \begin{pmatrix} a_{11}\hat{}b_{11} & a_{12}\hat{}b_{12} \\ a_{21}\hat{}b_{21} & a_{22}\hat{}b_{22} \end{pmatrix}.$$

Jeżeli s jest skalarem, to możliwe są operacje

$$s./A = \begin{pmatrix} s/a_{11} & s/a_{12} \\ s/a_{21} & s/a_{22} \end{pmatrix},$$

$$s.\hat{}A = \begin{pmatrix} s\hat{}a_{11} & s\hat{}a_{12} \\ s\hat{}a_{21} & s\hat{}a_{22} \end{pmatrix}.$$

1.23 Sumowanie

Jeżeli $x = [x_1, x_2, \ldots, x_n]$ jest wektorem, to funkcja

$$sum(x)$$

wykonuje sumowanie $\sum_1^n x_i$, natomiast funkcja

$$cumsum(x)$$

produkuje wektor

$$\left(\sum_1^1 x_i, \sum_1^2 x_i, \ldots, \sum_1^n x_i\right).$$

Gdy argumentem tych funkcji jest macierz

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix},$$

sum generuje wektor o składowych

$$\left(\sum_{i=1}^m a_{i1}, \sum_{i=1}^m a_{i2}, \ldots, \sum_{i=1}^m a_{in}\right),$$

czyli sumowane są kolumny, a *cumsum* macierz

$$\begin{pmatrix} \sum_{i=1}^1 a_{i1} & \sum_{i=1}^1 a_{i2} & \ldots & \sum_{i=1}^1 a_{in} \\ \sum_{i=1}^2 a_{i1} & \sum_{i=1}^2 a_{i2} & \ldots & \sum_{i=1}^2 a_{in} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^m a_{i1} & \sum_{i=1}^m a_{i2} & \ldots & \sum_{i=1}^n a_{in} \end{pmatrix}.$$

1.24 Działania matematyczne na macierzach

1. Dodawanie i odejmownie, p. podrozdział 1.22.

2. Mnożenie macierzowe. Jeżeli mamy dwie macierze A i B o wymiarach $(m \times p)$ i $(p \times n)$, odpowiednio, to polecenie

$$C = A * B$$

utworzy macierz C o wymiarze $(m \times n)$ i elementach

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{k,j},$$

gdzie $i = 1, \ldots, m$ i $j = 1, \ldots, m$ oraz a_{ij}, b_{ij} i c_{ij} są odpowiednio elementami macierzy A, B i C.

3. Iloczyn skalarny wektorów – realizowany w MATLAB-ie jako iloczyn wektora wierszowego $x = [x_1, x_2, \ldots, x_n]$ i wektor kolumnowego $y = [y_1, y_2, \ldots, y_n]'$, tzn.

$$x * y$$

lub poleceniem

$$dot(x, y)$$

4. Iloczyn tensorowy wektorów – realizowany w MATLAB-ie jako iloczyn wektora kolumnowego $x = [x_1, x_2, \ldots, x_m]'$ i wektora wierszowego $y = [y_1, y_2, \ldots, y_n]$, tzn.

$$A = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} \begin{pmatrix} y_1 & y_2 & \ldots & y_m \end{pmatrix} = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \ldots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \ldots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \ldots & x_m y_n \end{pmatrix},$$

czyli powstaje macierz A o wymiarze $(m \times n)$ i elementach $a_{ij} = x_i y_j$.

5. Obliczanie wyznacznika macierzy A – komenda

$$det(A)$$

6. Obliczanie wartości własnych macierzy A – polecenie

$$lambda = eig(A)$$

gdzie *lambda* jest wektorem kolumnowym zawierającym wartości własne macierzy A.

7. Odwracanie macierzy A – komenda

$$inv(A)$$

8. Obliczanie rzędu macierzy A (liczby liniowo niezależnych wierszy lub kolumn macierzy A) – polecenie

$$rank(A)$$

1.25 Rozwiązywanie układów równań liniowych

Rozważmy układ n równań liniowych z n niewiadomymi

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots &= \ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= b_n \end{aligned}$$

Możemy przepisać go w postaci macierzowej

$$Ax = b, \tag{1.1}$$

gdzie A jest macierzą kwadratową $(n \times n)$

$$A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix},$$

a x i b są n-elementowymi wektorami kolumnowymi

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{i} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}.$$

W MATLAB-ie rozwiązanie takiego układu można otrzymać używając poleceń

$$x = A\backslash b$$

lub

$$x = linsolve(A, b)$$

Lewy ukośnik oznacza tzw. lewe dzielenie macierzowe.

Rozdział 2

Numeryczna algebra liniowa i wyznaczanie zer funkcji

2.1 Eliminacja Gaussa

Rozważmy macierz dołączoną układu równań (1.1) po pewnym kroku procesu eliminacji

$$\left(\begin{array}{ccccccccc|c}
a_{11} & a_{12} & a_{13} & \dots & a_{1k} & \dots & a_{1j} & \dots & a_{1n} & b_1 \\
0 & a_{22} & a_{23} & \dots & a_{2k} & \dots & a_{2j} & \dots & a_{2n} & b_2 \\
0 & 0 & a_{33} & \dots & a_{3k} & \dots & a_{3j} & \dots & a_{3n} & b_3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \dots & a_{kk} & \dots & a_{kj} & \dots & a_{kn} & b_k \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \dots & a_{ik} & \dots & a_{ij} & \dots & a_{in} & b_i \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & \dots & a_{nk} & \dots & a_{nj} & \dots & a_{nn} & b_n
\end{array}\right)$$

wiersz pomocniczy

wiersz transformowany

Pierwszych k wierszy zostało już przetransformowanych do postaci trójkątnej górnej. Oznacza to, że teraz k-ty wiersz bedzie używany do transformacji następnych. Załóżmy, że chcemy przetransformować i-y wiersz. Uzyskamy to mnożąc k-ty wiersz przez $\lambda = a_{ik}/a_{kk}$ i odejmując go od wiersza i-tego, tzn.

$$a_{ij} - \lambda a_{kj} \to a_{ij}, \qquad b_i - \lambda b_k \to b_i$$

gdzie $j = k, k+1, \dots, n$.

Aby przetransformować całą macierz do postaci trójkątnej górnej k musi w tych formułach zmieniać się od 1 do $n-1$ (indeks wiersza używanego do transformacji), a i od $k+1$ do n (indeks wiersza transformowanego).

Po zakończeniu procesu eliminacji otrzymamy

$$\begin{pmatrix} a'_{11} & a'_{12} & a'_{13} & \dots & a'_{1n} \\ 0 & a'_{22} & a'_{23} & \dots & a'_{2n} \\ 0 & 0 & a'_{33} & \dots & a'_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & a'_{nn} \end{pmatrix} \begin{pmatrix} b'_1 \\ b'_2 \\ b'_3 \\ \vdots \\ b'_n \end{pmatrix}.$$

Najpierw rozwiązujemy ostatnie równanie

$$a'_{nn} x_n = b'_n$$

otrzymując

$$x_n = b'_n / a'_{nn}, \tag{2.1}$$

a pozostałe niewiadome dostajemy używając formuły (tzw. podstawienie w tył)

$$x_k = \left(b'_k - \sum_{j=k+1}^{n} a'_{kj} x_j \right) \Big/ a'_{kk}, \tag{2.2}$$

gdzie $k = n-1, n-2, \dots, 1$.

Element diagonalny jest nazywany elementem głównym lub z ang. *pivotem*. Opisana wyżej procedura eliminacji zawodzi, jeżeli *pivot* używany do eliminacji jest równy 0. Zmieniając miejscami równania (wiersze) lub zmienne (kolumny) przed każdym etapem eliminacji (tzw. z ang. *pivoting*), tak aby największy co do modułu element znalazł się na diagonali, unikniemy tego problemu. Taka procedura redukuje również błędy zaokrąglenia, ponieważ dzielenie przez duże liczby jest obarczone mniejszym błędem zaokrąglenia niż dzielenie przez małe liczby.

Wariacją eliminacji Gaussa jest eliminacja Gaussa-Jordana, w której elementy nad i pod diagonalą są eliminowane. W wyniku macierz A jest transformowana w macierz diagonalną.

Wady metody eliminacji Gaussa i jej wariantów:

- błędy zaokragleń, które mogą być zminimalizowane przez użycie dużej precyzji i *pivoting*;
- problemy z układami źle uwarunkowanymi:
  - problem *dobrze uwarunkowany*, to taki, w którym mała zmiana dowolnego elementu powoduje małe zmiany rozwiązania;
  - problem *źle uwarunkowany*, to taki, w którym mała zmiana dowolnego elementu powoduje duże zmiany rozwiązania, a zatem takie problemy są bardzo czułe na błędy zaokrągleń.

2.2 Norma i wskaźnik uwarunkowania

Norma jest miarą wielkości skalara a, wektora x lub macierzy A. Oznacza się ją symbolem $\|\cdot\|$ i ma następujące własności:

$$\begin{aligned} \|x\| &\geqslant 0, && (2.3)\\ \|x\| &= 0 \Leftrightarrow x = 0, && (2.4)\\ \|ax\| &= |a|\|x\|, && (2.5)\\ \|x+y\| &\leqslant \|x\| + \|y\|. && (2.6) \end{aligned}$$

Dodatkowo norma macierzy spełnia nierówność

$$\|AB\| \leqslant \|A\|\|B\|. \tag{2.7}$$

Normą skalara jest jego moduł, tzn. $\|a\| = |a|$.

Używa się kilku definicji normy wektora x:

- p-norma (polecenie MATLAB-a: $norm(x,p)$)

$$\|x\|_p = \left(\sum_{i=1}^{n} |x|^p\right)^{1/p}, \qquad 1 \leqslant p < \infty; \tag{2.8}$$

- komenda $norm(x,inf)$ oblicza normę

$$\|x\|_\infty = \max_{1\leqslant i\leqslant n} |x_i|\,; \tag{2.9}$$

- polecenie $norm(x,-inf)$ oblicza

$$\min_{1\leqslant i\leqslant n} |x_i| \tag{2.10}$$

i normy macierzy

- $norm(A,1)$, gdzie A jest macierzą $(m\times n)$, oblicza sumy odpowiednich kolumn macierzy i wybiera najwiekszą

$$\|A\|_1 = \max_{1\leqslant j\leqslant n} \sum_{i=1}^{m} |a_{ij}|; \tag{2.11}$$

- $norm(A,2)$, znajduje największą singularną wartość macierzy A, tzn, $max(svd(A))$[1];

[1] $[U,S,V] = svd(A)$ daje macierz diagonlną S o tym samym wymiarze co macierz A z nieujemnymi elementami diagonalnymi w porządku malejącym i macierze unitarne U i V, takie że $A = U * S * V'$; $S = svd(A)$ produkuje wektor zawierający wartości singularne

- $norm(A, inf)$, oblicza sumy odpowiednich wierszy macierzy i wybiera największą

$$\|A\|_\infty = \max_{1 \leqslant i \leqslant m} \sum_{j=1}^{n} |a_{ij}|; \tag{2.12}$$

- norma Frobeniusa ($norm(A,' fro')$)

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \right)^{1/2}. \tag{2.13}$$

2.3 Wskaźnik uwarunkowania macierzy

Wskaźnik uwarunkowania macierzy – układu równań jest miarą jego wrażliwości na małe zmiany jego elementów. Rozważmy układ

$$Ax = b. \tag{2.14}$$

Z własności normy macierzy mamy

$$\|b\| \leqslant \|A\| \, \|x\|. \tag{2.15}$$

Jeżeli zmienimy wektor b dodając do niego wektor Δb, to również zmieni się rozwiązanie równania (2.14)

$$A(x + \Delta x) = b + \Delta b. \tag{2.16}$$

Odejmując (2.14) od (2.16) dostajemy równanie

$$A\Delta x = \Delta b, \tag{2.17}$$

którego rozwiązaniem jest

$$\Delta x = A^{-1} \Delta b. \tag{2.18}$$

Obliczamy normy obu stron tego równania i korzystamy z własności normy, by otrzymać nierówność

$$\|\Delta x\| \leqslant \|A^{-1}\| \, \|\Delta b\|. \tag{2.19}$$

Mnożenie stronami tej nierówności przez nierówność (2.15) daje

$$\|b\| \, \|\Delta x\| \leqslant \|A\| \, \|x\| \, \|A^{-1}\| \, \|\Delta b\|. \tag{2.20}$$

Po podzieleniu obu stron tej nierówności przez $|b\| \, \|x\|$ otrzymamy

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \|A\| \, \|A^{-1}\| \, \frac{\|\Delta b\|}{\|b\|}. \tag{2.21}$$

Jak widać wrażliwość rozwiązania x na zmiany wektora b jest określona przez wartość

$$C(A) = \|A\| \, \|A^{-1}\|, \tag{2.22}$$

którą nazywamy wskaźnikiem uwarunkowania. Małe wartości $C(A)$, rzędu jedności oznaczają słabą wrażliwość na zmiany b – problem jest dobrze uwarunkowany. Gdy wskaźnik ten jest duży, problem jest źle uwarunkowany. Określenie „duży" zależy od kontekstu, ale na pewno wskaźnik uwarunkowania rzędu $1/eps$ jest duży, bo oznacza, że macierz A z dokładnością eps jest macierzą osobliwą[2].

Można pokazać, że zaburzenie macierzy A prowadzi do nierówności

$$\frac{\|\Delta x\|}{\|x + \Delta x\|} \leqslant C(A) \, \frac{\|\Delta A\|}{\|A\|}. \tag{2.23}$$

W MATLAB-ie wskaźnik uwarunkowania obliczany jest przez funkcję $cond(A, p)$, gdzie $p = 1, 2, inf,' fro'$. Użycie $cond(A)$ oznacza przyjęcie $p = 2$. Jednak obliczanie dokładnego wskaźnika uwarunkowania jest „drogie" – wymaga dużo czasu CPU. Z tego powodu MATLAB udostępnia funkcje *rcond* i *condtest*, które szacują prawidłowy rząd tego wskaźnika za trzykrotnie niższą cenę. Polecenie *rcond* oblicza przybliżoną wartość odwrotności tego wskaźnika.

2.4 Rozkład LU

Dowolna kwadratowa macierz A może być zapisana jako iloczyn macierzy trójkątnej dolnej L i trójkątnej górnej U

$$A = LU.$$

A zatem układ równań $Ax = b$ można przepisać jako $LUx = b$, a po wprowadzeniu pomocniczego wektora $y = Ux$ jako $Ly = b$. Ten ostatni układ możemy rozwiązać względem y metodą w podstawinia w przód. Mając y rozwiązujemy $Ux = y$ metodą podstawienia w tył.

Macierz U jest po prostu macierzą trójkątną górną otrzymaną metodą eliminacji Gaussa

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}. \tag{2.24}$$

Jeżeli założymy, że macierz trójkątna dolna ma na diagonali jedynki, tzn.

$$L = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix},$$

[2] w MATLAB-ie *eps* oznacza odległość od 1,0 do następnej większej liczby podwójnej precyzji, tzn. 2^{-52}

to

$$A = LU = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{11}l_{21} & u_{12}l_{21} + u_{22} & u_{13}l_{21} + u_{23} \\ u_{11}l_{31} & u_{12}l_{31} + u_{22}l_{32} & u_{13}l_{31} + u_{23}l_{32} + u_{33} \end{pmatrix}.$$

Do tej macierzy zastosujemy eliminację Guassa. Zaczynamy od użycia pierwszego wiersza do zerowania elementów poddiagonalnych pierwszej kolumny. Mnożymy go przez element l_{31} i odejmujemy go od wiersza trzeciego, potem mnożymy go przez l_{21} i odejmujemy od wiersza drugiego. Po wstawieniu wyników, odpowiednio, za wiersz trzeci i drugi dostajemy

$$A_1 = LU = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & u_{22}l_{32} & u_{23}l_{32} + u_{33} \end{pmatrix}.$$

W następnym kroku używamy wiersza drugiego do zerowania elementów poddiagonalnych drugiej kolumny. W wyniku tej operacji otrzymamy macierz trójkątną górną (2.24).

Na tym prostym przykładzie pokazaliśmy dwie ważne cechy metody rozkładu LU Doolitlle'a, a mianowicie:

1. Macierz U jest identyczna z macierzą trójkątną górną, będącą wynikiem eliminacji Gaussa;

2. Pozadiagonalne elementy macierzy L są czynnikami używanymi w eliminacji Gaussa – mnożnik l_{ij} jest używanu do elimincji elementu a_{ij}.

W MATLAB-ie rozkładu LU macierzy A można dokonać poleceniem $lu(A)$.

2.5 Podstawienie w przód

Rozważmy procedurę rozwiązywania równania $Ly = b$. Po rozpisaniu układ równań ma postać

$$\begin{aligned} y_1 &= b_1 \\ l_{21}y_1 + y_2 &= b_2 \\ &\vdots \\ l_{k1}y_1 + l_{k2}y_2 + \ldots + l_{k,k-1}y_{k-1} + y_k &= b_k \\ &\vdots \end{aligned}$$

Rozwiązując k-te równanie dostajemy

$$y_k = b_k - \sum_{j=1}^{k-1} l_{kj}, \qquad k = 2, 3, \ldots, n. \tag{2.25}$$

Układ równań $Ux = y$ rozwiązujemy algorytmem podstawienia w tył (2.1-2.2).

2.6 Rozkład Choleskiego

Jeżeli macierz A jest symetryczna (hermitowska) i dodatnio określona, to można ja przedstawić w postaci $A = LL^T$ ($A = LL^+$). Dla macierzy symetrycznej (3×3) rozkład Choleskiego zapisujemy następująco

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{pmatrix}. \tag{2.26}$$

Po wymnożeniu otrzymujemy

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{32} \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{pmatrix}. \tag{2.27}$$

Ponieważ macierz A jest symetryczna, przyrównanie odpowiednich elementów obu macierzy da sześć równań, w których niewiadomymi są elementy macierzy L. Rozpoczynamy od pierwszych kolumn i otrzymujemy

$$\begin{aligned} a_{11} = l_{11}^2 &\Rightarrow l_{11} = \sqrt{a_{11}}, \\ a_{21} = l_{11}l_{21} &\Rightarrow l_{21} = a_{21}/l_{11}, \\ a_{31} = l_{11}l_{31} &\Rightarrow l_{31} = a_{31}/l_{11}. \end{aligned}$$

Używając drugich kolumn począwszy od drugiego wiersza dostajemy

$$\begin{aligned} a_{22} = l_{21}^2 + l_{22}^2 &\Rightarrow l_{22} = \sqrt{a_{22} - l_{21}^2}, \\ a_{32} = l_{21}l_{31} + l_{22}l_{32} &\Rightarrow l_{32} = (a_{32} - l_{21}l_{31})/l_{22}. \end{aligned}$$

I na koniec dla elemntów z $i = j = 3$ otrzymujemy

$$a_{33} = l_{31}^2 + l_{32}^2 + l_{33}^2 \Rightarrow l_{33} = \sqrt{a_{33} - l_{31}^2 - l_{32}^2}.$$

Te wyniki możemy uogólnić dla macierzy $(n \times n)$. Zauważmy, że otrzymane równania można zapisać w postaci

$$a_{ij} = \sum_{k=1}^{j} l_{ik}l_{jk}, \quad i = j, j+1, \ldots, n, \quad j = 1, 2, \ldots, n. \tag{2.28}$$

Jeżeli zaczniemy obliczenia od pierwszej kolumny (tzn. od $j = 1$) i bedziemy zmieniać i od 1 do n, a potem przyjdziemy w ten sam sposób po kolei do następnych kolumn, to niewiadomą w każdym z otrzymanych z (2.28) równań

bedzie element l_{ij} (inne elementy macierzy L, występujące w tym równaniu, zostały już obliczne z poprzednio wygenerownych rownań). Wyłączając ten element spod znaku sumy otrzymujemy z (2.28)

$$a_{ij} = \sum_{k=1}^{j-1} l_{ik} l_{jk} + l_{ij} l_{jj}. \tag{2.29}$$

Jeżeli $j = 1$, to

$$l_{11} = \sqrt{a_{11}} \tag{2.30}$$

$$l_{i1} = a_{i1}/l_{11}, \quad i = 2, 3, \ldots, n. \tag{2.31}$$

Gdy $i = j$ (elementy diagonalne), to

$$l_{jj} = \sqrt{a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2} \quad j = 2, 3, ..., n, \tag{2.32}$$

i dla elemntów niediagonalnych ($i \neq j$)

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) / l_{jj} \quad j = 2, 3, ..., n-1, \quad i = j+1, j+2, \ldots, n. \tag{2.33}$$

W MATLAB-ie rozkładu Choleskiego macierzy A można dokonać poleceniem $chol(A)$.

2.7 Układy liniowe w MATLAB-e

Podstawowym narzędziem MATLAB-a do rozwiązywania układów liniowych jest operator „\". Można go stosować do trzech typów układów liniowych $Ax = b$, gdzie, macierz A i wektor b są zadane, a mianiowicie do układów kwadratowych, podokreślonych i nadokreślonych. W ogólności ten operator może być użyty do rozwiązania wielu systemów zapisanych jako $AX = B$, gdzie B jest macierzą k-kolumnową – w takim wypadku rozwiązywane po kolei bedą układy $AX(:, j) = B(:, j)$ dla $j = 1, \ldots, k$. Inaczaj mówiąc polecenie $X = A\backslash B$ wygeneruje macierz X, której kolumny bedą rozwiązaniami układów określonych macierzą A i odpowiednimi kolumnami macierzy B.

Gdy macierz A jest kwadratowa i nieosobliwa, to komenda $A\backslash b$ powoduje wykonanie rozkładu LU macierzy A z częściowym *pivotingiem*. Jednocześnie jest obliczana funkcja $rcond(A)$ i jeżeli jej wartość jest mniejsza od *eps* wyświetlany jest odpowiedni komunikat.

MATLAB rozpoznaje trzy specjalne postacie macierzy kwadratowych i używa dedykowanych procedur do ich rozwiązania:

1. Macierz trójkątna lub jej permutacja – układ jest rozwiązywany przez podstawienie;

2. Macierz Hessenberga (jeżeli $a_{ij} = 0$ dla $i > j+1$, to macierz jest górną macierzą Hessenberga, tzn. jest macierzą trojkatną górną z niezerową pierwszą poddiagonalą) – stosowany jest rozkład LU przeznaczony dla tego typu macierzy;

3. Macierz hermitowska dodatnio określona – używany jest rozkład Choleskiego.

Jeżeli zależy nam na wydajności obliczeń i macierz A ma odpowiednią strukturę, to zamiast operatora „\" bardziej właściwe będzie użycie polecenie $linsolve(A, B, opts)$, które rozwiązuje układy $AX = B$ ($A^*X = B$, jeżeli opcja $opts.TRANSA$ ma wartość $true$). Struktura $opts$ pozwala wyspecyfikowć jedną z kilku możliwych własności macierzy A, więc badanie własności tej macierzy, które wykonuje operator „\", nie jest potrzebne, co zwiększa efektywność obliczeń. Np. jeżeli macierz A jest macierzą trójkatną górną, to powinniśmy przypisać opcji $opts.UT$ wartość $true$ poleceniem

$$opts.UT = true$$

i wywołać

$$X = linsolve(A, B, opts)$$

W tabeli 2.1 zamieszczono wszystkie nazwy pól struktury $opts$ i odpowiadające im własności macierzy. Wartości tych pól mogą przyjmować wartości logiczne: $true$ lub $false$ (z założenia wartości tych pól są położone równe $false$).

Tabela 2.1: Pola struktury $opts$ i odpowiadające im własności macierzy

Nazwa pola	Własność macierzy
LT	trójkątna dolna
UT	trójkątna górna
$UHESS$	Hessenberga górna
SYM	rzeczywista symetryczna lub zespolona hermitowska
$POSDEF$	dodatnio określona
$RECT$	ogólna prostokątna
$TTRNSA$	$false$ – $A * X = B$, $true$ – $A' * X = B$

Jeżeli np. macierz A jest macierzą trójkątną górną, to kładziemy

$$opts.UT = true$$

Ponieważ $linsolve$ nie sprawdza, czy własności macierzy zostały prawidłowo wyspecyfikowane, produkuje złe wyniki bez żadnego ostrzeżenia.

2.8 Równania nieliniowe

Procedurę wyznaczania pierwiastków rownania nieliniowego $f(x) = 0$ można podzielić na dwa etapy: lokalizację rozwiązania i obliczanie coraz to lepszego jego przybliżenia. Lokalizacja pierwiastka polega na jego zgrubnym oszczacowaniu, np. przez wyznaczenie przedziału (a, b), w którym funkcja nieliniowa zmienia znak, tzn. $f(a)f(b) < 0$. Można odczytać te wartości z wykresu lub okreslić duży obszar, w którym spodziewamy się miejsc zerowych, a potem obliczać wartość funkcji $f(x)$ wewnątrz tego obszaru zwiększając po kolei x o małą wartość Δx począwszy od lewego krańca tego obszaru i w ten sposób znaleźć przedziały $(x_i, x_i + \Delta x)$ zawierające pierwiastki. Jeżeli podejrzewamy, że w takim przedziale występują pierwiastki wielokrotne, powinniśny zbadać jak zmnienia się znak pochodnej $f'(x)$ w tym przedziale.

Drugi etap polega na systematycznym generowaniu kolejnych przybliżeń pierwiastka, aż jego wartość spełni założone warunki dokładności. Metody generowania takich przybliżeń możemy podzielić na dwie kategorie: wymagające, aby pierwiastek zawsze znajdował się w coraz to mniejszym przedziale (np. metoda bisekcji i metoda $regula\ falsi$) i niewymagające zamykania pierwiastka w przedziale (np. metoda iteracji prostej, metoda siecznych i metoda Newtona). Te ostatnie metody nie muszą być zbieżne, ale są one dużo bardziej efektywne.

Algorytmy obliczajace pierwiastki powinny uwzględniać:

- górną granicę na liczbę iteracji;
- testownie zbieżności: $|x_{i+1} - x_i| < \epsilon_1$ lub $|f(x_{i+1})| < \epsilon_2$;
- badanie, czy $f'(x)$ nie osiąga zera (w wypadku metod używających wartości pochodnej).

2.9 Metoda bisekcji

Jeżeli znamy przedział (a, b), w którym leży pierwiastek, to możemy go podzielić na dwa podprzedziały uśredniając a i b, by otrzymać punkt podziału $c = (a + b)/2$. Mamy teraz dwa przedziały (a, c) i (c, b). Przedział, który zawiera pierwiastek jest określony przez wartość $f(c)$. Jeżeli $f(a)f(c) < 0$, to pierwiastek jest w przedziale (a, c) i kładąc $b = c$ możemy kontynuować proces bisekcji. W przciwnym razie, gdy $f(a)f(c) > 0$ pierwiastek znajduje się w przedziale (c, b) i kładąc teraz $a = c$ możemy powtórzyc dzielenie przedziału. Jeżeli $f(a)f(c) = 0$, to c jest pierwiastkiem i kończymy iterację. Podsumowując, otrzymujemy następujący algorytm:

1. Obliczamy $c = (a + b)/2$;
2. Jeżeli $|f(c)| < \epsilon_2$ lub $|b - a| < \epsilon_3$, to pierwiastek równa się c; koniec algorytmu;

3. Jeżeli $f(a)f(c) < 0$, to $b = c$; powtarzamy algorytm;

4. Jeżeli $f(a)f(c) > 0$, to $a = c$; powtarzamy algorytm.

Ten algorytm jest algorytmem iteracyjnym, a iteracje są kontnuowane aż wielkość przedziału stanie się mniejsza od zadanej tolerancji ϵ_3 lub wartość bezwzględna funkcji nie zmniejszy się poniżej założonej tolerancji ϵ_2.

Jeżeli równanie nieliniowe ma osobliwość, jak np. $f(x) = 1/(x - d)$ w punkcie $x = d$, bisekcja zlokalizuje tę nieciągłość i wtedy $|f(x)|$ dla $x \to d$ ją zidentyfikuje.

Zalety metody bisekcji:

- zawsze zbieżna;
- można oszacować maksymalny błąd pierwiastka, który jest równy $|b_n - a_n|/2$, gdzie $n = 0, 1, \ldots$ oznacza numer iteracji;
- można obliczyć liczbę iteracji konieczną do zmniejszenia długości początkowego przedziału $(b_0 - a_0)$ do $(b_n - a_n)$; jest ona dana przez równanie
$$b_n - a_n = (b_0 - a_0)/2^n.$$

Główną wadą metody bisekcji jest jej wolna zbieżność, co oznacza dużą liczbę itercji i co za tym idzie dużą liczbę obliczeń wartości funkcji $f(x)$.

2.10 Metoda *regula falsi*

W metodzie *regula falsi* w odróżnieniu od metody bisekcji, w która przyjmuje się jako pierwiastek w przedziale (a, b) środek tego przedziału, zakłada się, że nieliniowa funkcja $f(x)$ jest tym przedziale liniową funkcją $g(x)$ i pierwiastek tej funkcji c $(g(c) = 0)$ jest przyjmowany za przybliżenie pierwiastka funkcji $f(x)$. Mamy teraz dwa podprzedziały (a, c) i (c, b). Tak jak w metodzie bisekcji wybieramy ten, który zawiera pierwiastek.

Prosta $g(x)$ przechodzi przez trzy punkty $(a, f(a))$, $(b, f(b))$ i $(c, 0)$. Jej współczynnik kierunkowy $g'(x)$ (tagens kąta nachylenia) jest zatem równy

$$g'(x) = \frac{f(b) - f(a)}{b - a} = \frac{f(b) - 0}{b - c} = \frac{0 - f(a)}{c - a}.$$

Rozwiazując te równania względem c dostajemy

$$c = b - f(b)\frac{b - a}{f(b) - f(a)}$$

lub

$$c = a - f(a)\frac{b - a}{f(b) - f(a)}.$$

Podobnie jak w wypadku bisekcji iterujemy ten algorytm aż osiągnięta zostanie zadana dokładność. Metoda *regula falsi* jest szybciej zbieżna niż metoda bisekcji.

2.11 Metoda punktu stałego

W algorytmie nazywanym metodą punktu stałego rozwiązujemy równanie $f(x) = 0$ przekształcając je najpierw do postaci $x = g(x)$. Potem znajdujemy $x = a$, takie że $a = g(a)$, używając schematu iteracyjnego

$$x_{i+1} = g(x_i), \tag{2.34}$$

aż zostanie spełniony zadany warunek zbieżności.

Zabdajmy kiedy proces iteracyjny (2.34) jest zbieżny. Jeżeli $e_i = (x_i - a)$ oznacza błąd i-tego przybliżenia pierwiastka, to

$$e_{i+1} = x_{i+1} - a = g(x_i) - g(a), \tag{2.35}$$

gdzie skorzystaliśmy z (2.34). Używając rozwinięcia Taylora w otoczeniu punktu x_i dostajemy

$$g(a) = g(x_i + a - x_i) = g(x_i) + g'(\eta)(a - x_i), \tag{2.36}$$

gdzie $\eta \in [x_i, a]$. Wyznaczamy z tego równania $g(x_i) - g(a)$ i wstawiamy do (2.35), co daje

$$e_{i+1} = g'(\eta)e_i. \tag{2.37}$$

Mówimy, że metoda iteracyjna jest zbieżna, jeżeli bląd absolutny maleje, tzn. gdy $|e_{i+1}/e_i| < 1$. W przypadku metody iteracji prostej oznacza to, że powinien być spełniony warunek $|g'(\eta)| < 1$. Zbieżność tej metody jest liniowa, bo e_{i+1} zależy liniowo od e_i.

2.12 Metoda Newtona-Raphsona

Załóżmy, że mamy przybliżenie x_i szukanego pierwiastka równania $f(x) = 0$. Przybliżamy lokalnie funkcję nieliniową $f(x)$ liniową funkcją $g(x)$ styczną do funkcji $f(x)$ w punkcie $(x_i, f(x_i))$. Jako następne przybliżenie piewiastka x_{i+1} przyjmujemy punkt przecięcia tej stycznej z osią OX. Ponieważ jej współczynkiem kierunkowym jest $f'(x)$ i przechodzi ona przez punkt $(x_i, f(x_i))$, dostajemy

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{2.38}$$

Zbadajmy zbieżność tej metody, która jest nazywana metodą Newtona-Raphsona. Korzystając z (2.38) możemy napisać

$$x_{i+1} = g(x_i), \tag{2.39}$$

czyli tak jak w wypadku metody iteracji prostej proces iteracyjny będzie zbieżny (por. (2.37), gdy

$$|g'(\eta)| = \left| \frac{f(x)f''(x)}{[f'(x)]^2} \right| < 1, \tag{2.40}$$

gdzie η leży między x_i i wartością pierwiastka a.

Niech $e_i = x_i - a$ będzie błędem i-go przybliżenia pierwiastka. Wtedy z (2.38) mamy

$$e_{i+1} = e_i - \frac{f(x_i)}{f'(x_i)}, \tag{2.41}$$

a ze wzoru Taylora

$$f(a) = f(x_i) + f'(x_i)(a - x_i) + \frac{1}{2}f''(\eta)(a - x_i)^2 = 0. \tag{2.42}$$

Z tego ostatniego wzoru wyznaczamy

$$f(x_i) = f'(x_i)e_i + \frac{1}{2}f''(\eta)e_i^2, \tag{2.43}$$

wstawiamy do (2.41) i końcu otrzymujemy

$$e_{i+1} = \frac{1}{2}\frac{f''(\eta)}{f'(\eta)}e_i^2, \tag{2.44}$$

czyli metoda Newtona jest zbieżna kwadratowo. Jest to zaleta tej metody. Wadą jest konieczność obliczania pochodnej $f'(x)$.

2.13 Metoda siecznych

Załóżmy, że mamy dwa przyliżenia x_{i-1} i x_i pierwiastka a równania $f(x) = 0$. Tę nieliniową funkcję przybliżamy sieczną $g(x)$ przechodzącą przez punkty $(x_{i-1}, f(x_{i-1}))$ i $(x_i, f(x_i))$. Miejsce zerowe tej prostej x_{i+1} przyjmujemy jako następne przybliżenie pierwiastka.

Prosta $g(x)$ przechodzi przez trzy punkty $(x_{i-1}, f(x_{i-1}))$, $(x_i, f(x_i))$ i $(x_{i+1}, 0)$. Jej współczynnik kierunkowy $g'(x)$ jest zatem równy

$$g'(x) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} = \frac{0 - f(x_i)}{x_{i+1} - x_i}.$$

Rozwiazując to równanie względem x_{i+1} dostajemy

$$x_{i+1} = x_i - f(x_i)\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}.$$

Szybkość zbieżności tej metody jest dana przez

$$e_{i+1} = \left[\frac{1}{2}\frac{f''(\eta)}{f'(\eta)}\right]^{0.62} e_i^{1.62}. \tag{2.45}$$

2.14 Tworzenie funkcji w MATLAB-ie

Jedną z form m-plików MATLAB-a są pliki zawierające funkcje. Są to skrypty tworzące swoją lokalną przestrzeń roboczą w ramach środowiska MATLAB-a. Zmienne zdefiniowane wewnątrz funkcji mają charakter lokalny.

Funkcja składa się z co najmniej dwóch linii kodu. Pierwsza musi mieć format wymagany przez MATLAB-a. Jest to tzw. interface funkcji, który ma postać

$$function[\text{zmienne wyjściowe}]=\text{Nazwafunkcji}(\text{zmienne wejściowe}),$$

gdzie „zmienne wyjściowe" oznaczają listę zmiennych wyjściowych oddzielonych przecinkami, a „zmienne wejściowe" analogiczną listę zmiennych wejściowych. „Nazwafunkcji" jest nazwą funkcji i jednocześnie nazwą m-pliku, która musi zaczynać sie od litery. Aby funkcja była dostępna w czasie sesji, plik ją zawierający powinien znajdować się w bieżącym folderze lub w folderze, który jest zdefiniowany jako folder z funkcjami, np. za pomocą odpowiedniego menu.

Komentarze umieszczone bezpośrednio po linii interface'u, są używane przez MATLAB-a do generacji informacji o tej funkcji (można ją wyświetlić używając polecenia *help* Nazwafunkcji).

Zasady stosowania funkcji w MATLAB-ie, nie różnią się w ogólności od zasad ich stosowania w innych językach programowania (np. zmienne stosowane w funkcjach mają charakter lokalny, chyba, że użyjemy instrukcji *global* do zdefiniowania zmiennych globalnych).

Funkcja przekazuje kontrolę do programu ją wywołującego po wykonaniu ostatniej instrukcji. Aby wymusić wcześniejszy taki powrót używamy polecenia *return*.

Jeżeli słowo kluczowe $function$ pojawia się w m-pliku więcej niż raz, dodatkowe funkcje nazywamy subfunkcjami. Są one dostępne tylko dla funkcji zdefiniowanych w tym pliku.

2.14.1 Funkcje anonimowe

Innym rodzajem funkcji, które można tworzyć w okienku *command*, w skryptach, w funkcjach i subfunkcjach są funkcje anonimowe. Taka funkcja składa się tylko z jednego wyrażenia i generuje tylko jedną zmienną. Definiowana jest w następujący sposób

$$\text{uchwytfunkcji}=@(\text{argumenty})(\text{wyrażenie}),$$

gdzie „uchwytfunkcji" jest uchwytem funkcji, „argumenty" oznaczają listę nazw zmiennych oddzielonych przecinkami, a „wyrażenie" jest akceptowanym przez MATLAB-a wyrażeniem (nawiasy zawierające wyrażenie nie są wymagane, ale ułatwiają czytanie kodu), np.

$$arka = @(x, y)(sqrt(x + y))$$

Wyrażenie definujące funkcję anonimową może zawierać inną funkcję anonimową.

2.14.2 Funkcje typu *inline*

Inną metodą tworzenia funkcji lokalnych jest użycie instrukcji *inline* w następujący sposób

$$\text{Nazwafunkcji}=inline(\text{'wyrażenie','z1','z2',}\ldots)$$

gdzie „wyrażenie" jest wyrażeniem akceptowanym przez MATLAB-a, a „z1", „z2", ... są nazwami zmiennych występujacych w „wyrażeniu". Na przykład

$$arkainline = inline('x + y','x','y')$$

Takie funkcje można kreować w okienku *command*, w skryptach, w funkcjach i subfunkcjach. Nie mogą one jednak używać innych funkcji typu *inline*, ale mogą przywoływać zwykłe funkcje, tzn. istniejące jako m-pliki.

2.14.3 Uchwyty funkcji i polecenie $feval$

Wiele funkcji MATLAB-w wymaga od użytkownika wykreowania funkcji w formie określonej przez składnię właściwą dla tych funkcji. Te funkcje stosują funkcję MATLAB-a

$$feval(\text{Uchwytfunkcji,p1,p2,}\ldots\text{,pn})$$

gdzie „Uchwytfunkcji" jest uchwytem funkcji, a „p1, p2, ..., pn" są parametrami przekazywanymi do funkcji reprezentowanej przez ten uchwyt.

Taką funkcją jest $fzero$, która znajduje numeryczne rozwiązanie równania $f(x) = 0$ z tolerancją t, w otoczeniu punktu x_0 lub w w przedziale określonym przez wektor $[x_1, x_2]$. Może ona również przekazać parametry p_i, $i = 1, 2, \ldots$, do funkcji definiującej $f(x)$. Używamy funkcji $fzero$ w następujący sposób

$$x = fzero(@Nazwafunkcji, x0, options, p1, p2, \ldots)$$

gdzie x jest przybliżeniem pierwiastka, tzn. $f(x) \approx 0$, $Nazwafunkcji$ jest uchwytem funkcji lub subfunkcji, $x0 = x_0$ lub $x0 = [x_1, x_2]$ oraz $p1$, $p2$, itd., są parametrami wymaganymi przez funkcję, której uchwytem jest $Nazwafunkcji$. Opcje *options* są określane przez polecenie *optimset*.

Interface funkcji używanej przez $fzero$ musi mieć formę

$$\begin{aligned} &function\ f = Nazwafunkcji(x, p1, p2, ..)\\ &instrukcje\\ &f = \ldots \end{aligned}$$

gdzie x jest zmienną niezależną, którę $fzero$ zmienia poszukując piewrwiastka.

Funkcja $fzero$ może również użyć funkcji anonimowej postaci

$$Uchwytfunkcji = @(x, p1, p2, ..)(wyrażenie)$$

w następujący sposób

$$x = fzero(Uchwytfunkcji, x0, options, p1, p2, \ldots)$$

lub funkcji typu *inline*

$$Inlinenazwafunkcji = inline('wyraznie', 'x', 'p1', 'p2', ..)$$

następująco

$$x = fzero(Inlinenazwafunkcji, x0, options, p1, p2, \ldots)$$

Algorytm stosowany przez $fzero$ jest kombinacją metod bisekcji, siecznych i odwrotnej interpolacji kwadratowej (p. „help" MATLAB-a).

Rozdział 3

Zasady prowadzenia obliczeń numerycznych – błędy i niestabilności

3.1 Operacje arytmetyczne w MATLAB-ie

Wszystkie operacje arytmetyczne w MATLAB-ie są prowadzone w arytmetyce zmiennoprzecinkowej podwójnej precyzji zgodnej ze standardem IEEE[1], tzn.:

- liczba zmiennoprzecinkowa jest reprezentowana jako

$$\pm c.cc\ldots c \times b^w,$$

  gdzie $c.cc\ldots c$ jest mantysą, $b = 2$ bazą systemu liczbowego i w wykładnikiem;

- liczba zajmuje 64 bity (8 bajtów);
- znak liczby określa 1 bit;
- na wykładnik przeznaczono 11 bitów;
- mantysa składa się z 52 bitów,
- minimalny wykładnik wynosi -1022;
- maksymalny wykładnik równa się +1023;
- przybliżona liczba dziesiętnych cyfr znaczących – 15,95;
- moduły liczb niezerowych zawierają się w przybliżeniu między 10^{-308} i 10^{+308};

[1]IEEE – skrót ang. nazwy *Institute of Electrical and Electronics Engineers*

- górna granica błędu względnego spowodowanego zaokrąglaniem wynosi $2^{-53} \approx 1,11 \times 10^{-16}$;
- odległość liczby 1,0 od najbliższej większej liczby zmiennoprzecinkowej wynosi $2^{-52} \approx 2,2204 \times 10^{-16}$ (można ją otrzymać używając polecenia MATLAB-a *eps*).

Każdy wynik obliczeń prowadzonych w MATLAB-ie jest zatem liczbą zmiennoprzecinkową podwójnej precyzji, chyba że wystąpi jeden z możliwych wyjątków, tzn.:

- jeżeli wynik obliczeń jest większy niż wartość zwracana przez funkcję $realmax/\text{-}realmax$, wtedy występuje przepełnienie i rezultatem jest $Inf/\text{-}Inf$, który reprezentuje nieskończoność/minus nieskończoność;
- jeżeli wynik obliczeń nie jest zdefiniowany matematycznie, to otrzymujemy NnN (ang. skrót od *not a number*); dotyczy to np. wyrażeń $0/0$, inf/inf, $0 * inf$, $inf - inf$, $NaN - NaN$ i $0 * NaN$;

Funkcja $realmin$ zwraca namniejszą dodatnią znormalizowaną liczbę zmiennoprzecinkową[2]. Jeżeli wynik obliczeń jest dodatni ale większy niż $eps{*}realmin \approx 4.9406 \times 10^{-324}$, to generowana jest liczba zdenormalizowana (z zerowymi początkowymi bitami) lub zero (niedomiar).

3.2 Cyfry znaczące

Cyfry znaczące liczby to te, o których wiadomo, że są poprawne. Zazwyczaj wiadomo ile cyfr znaczących mają dane wejściowe do obliczeń naukowych. Gdy te liczby są przetwarzane przez komputer i zaimplementowane w programach algorytmy numeryczne liczba cyfr znaczacych maleje, dlatego możliwość oszacowania liczby takich cyfr wyniku końcowego jest kluczowa dla oceny jakości otrzymanych rezultatów.

3.3 Precyzja i dokładność

Termin precyzja określa jakość reprezentacji liczby przez komputer. Inaczej mówiąc jak bliska jest liczba używana przez komputer liczby, którą ma reprezentować. Precyzja jest zdeterminowana przez liczbę cyfr (bitów) używanych w obliczeniach numerycznych, która wyznacza górną granicę błędu względnego spowodowanego zaokrąglaniem. Oczywiste jest, że liczby podwójnej precyzji są w ogólności bliższe liczbom rzeczywistym, które mają reprezentować niż liczby pojedynczej precyzji.

[2] najbardziej wysunięty w lewo bit mantysy znormalizowanej liczba zmiennoprzecinkowej jest zawsze równy 1

Dokładność mówi o tym jak dobrze liczba zgadza się z prawdziwą liczbą, którą reprezentuje, i jest określona przez błędy w przybliżeniach numerycznych.

3.4 Błędy związane z obliczeniami numerycznymi

Dokładność obliczeń numerycznych jest określona przez błędy tych obliczeń. W czasie takich obliczeń mogą wystapić następujące rodzaje błędów:

- błąd iteracji – pojawia się, gdy metoda iteracyjna osiąga dokładne rozwiązanie asymptotycznie; maleje z każdą wykonaną iteracją i może być zredukowany do granicy wyznaczonej przez możliwości komputera; przykładami takich błędów są błędy rozwiązań równań nieliniowych i układów równań liniowych, które otrzymano metodami iteracyjnymi (p. Rozdz. 2);
- błąd aproksymacji – jest róznicą między ścisłym rozwiązaniem problemu i ścisłym rozwiązaniem aproksymacji tego problemu; może być zredukowany przez użycie bardziej dokładnej aproksymacji; występuje np. gdy aproksymujemy funkcję wielomianem (iterpolacja, całkowanie numeryczne, itp.), gdy rozwiązując równanie różniczkowe aproksymujemy pochodne formułami różnicowymi obarczonymi tzw. błędami obcięcia;
- błędy zaokrąglenia – spowodowane przez skończoną długością słowa maszynowego[3] używanego w obliczeniach; są największe, gdy obliczane są małe różnice między dużymi liczbami; występują, gdy reprezentujemy liczbę dzisiętną liczbą binarną o skończonej długości i gdy wykonujemy operacje arytmetyczne na takich liczbach (wynik jest zazwyczaj dłuższą liczbą binarną niż ta, która może być reprezentowana na komputerze); błędy te akumulują się, tzn. rosną wraz z liczbą wykonanych operacji arytmetycznych.

3.5 Błąd absolutny i względny

Niech x_a będzie przybliżoną wartością liczby x, wtedy błędem bezwzględnym wartości x_a jest

$$\Delta x = |x_a - x| ,$$

a błędem względnym, o ile $x \neq 0$,

$$\frac{\Delta x}{|x|} = \left| \frac{x_a - x}{x} \right| .$$

[3] słowo maszynowe to elementarna porcja informacji, którą przetwarza system komputerowy – w przypadku maszyn binarnych jest określone przez z góry ustaloną liczbę bitów (zazwyczaj są to 32 lub 64bity)

Dokładność obliczeń jest określona przez te błędy.

Prezentacja wyników numerycznych powinna zawierać oszacowanie błędów, w przeciwnym razie tylko ostatnia cyfra dzisiętna zamieszczonej wartości jest zazwyczaj traktowana jako niepewna.

3.6 Matematyczna i numeryczna równoważność algorytmów

Dwa algorytmy są równoważne matematycznie, jeżeli dają dokładnie te same rezultaty dla tych samych danych wejściowych, gdy obliczenia są wykonywane bez błędów zaokrąglenia (inaczej mówiąc z nieskończoną liczbą cyfr). Z reguły jeden algorytm może być wyprowadzony z drugiego za pomocą operacji algebraicznych.

Algorytmy są równoważne numerycznie, jeżeli ich wyniki, otrzymane za pomocą arytmetyki zmiennoprzecinkowej są dla tych samych danych wejsciowych, są takie same.

Na przykład w obliczeniach numerycznych występuje często wyrażenie

$$c = \sqrt{a^2 + b^2},$$

którego użycie może prowadzić do wystąpienia przepełnienia lub niedomiaru, spowodowanych przez podnoszenie do kwadratu. Można tego uniknać przez zastosowanie równoważnego matematycznie algorytmu

$$c = \max(|a|, |b|)\sqrt{1 + \frac{\min(|a|, |b|)}{\max(|a|, |b|)}}, \quad \text{dla} \quad a \neq 0 \quad \text{lub} \quad b \neq 0,$$

i

$$c = 0, \quad \text{dla} \quad a = b = 0.$$

3.7 Olbrzymia utrata dokładności przy odejmowaniu bliskich liczb

Jednym z częstych powodów małej dokładności wyników obliczeń jest występowanie w nich odejmowania liczb, których różnica jest znacznie mniejsza od każdej z nich. Taka operacja generuje duży błąd względny. Nie dotyczy to dodawania, mnożenia i dzielenia takich liczb.

Rozważmy odejmowanie $z = x - y$, gdzie $x_a = x + \Delta x$ i $y_a = y + \Delta y$ są przybliżeniami dokładnych wartości. A zatem wynikiem odejmowania jest $z_a = z + \Delta z$, gdzie $\Delta z = \Delta x - \Delta y$. Ale ponieważ wartości Δx i Δy mogą mieć przeciwne znaki, to prawdziwe jest oszacowanie

$$|\Delta z| \leqslant |\Delta x| + |\Delta y|,$$

z którego wynika następujące oszacowanie błędu względnego

$$\frac{|\Delta z|}{|z|} \leqslant \frac{|\Delta x| + |\Delta y|}{|x - y|}.$$

Ponieważ błędy przybliżeń się sumują, to błąd względny odejmowania prawie równych liczb może być ogromny.

3.8 Stabilność algortymów

Jak pokazaliśmy w sekcji 2.3 istnieją problemy źle uwarunkowane, których rozwiąznie może być obarczone dużym błędem. Również źle skonstruowany algorytm może spowodować dużą utratę dokładności obliczeń. W ogólności mówimy, że algorytm jest niestabilny, jeżeli jego użycie może wprowadzić duże błędy do numerycznego rozwiązania dobrze uwarunkowanego problemu.

Relacje rekurencyjne pojawiają się dosyć często w różnych obliczeniach numerycznych. Stabilność takiej rlacji oznacza, że błąd pojawiający się na dowolnym etapie jest stosowania rośnie wolniej rośnie poszukiwane rozwiązanie.

3.9 Sumowanie szeregów nieskończonych

Jeżeli wyrazy sumowanego szeregu maleją dostatecznie szybko, można go urwać i przyjąć za oszacowanie błędu jego sumy ostatni uwzględniony lub pierwszy nieuwzględniony wyraz. Nie jest to precyzyjna reguła i w zasadzie określnie kryterium urywania szeregu wymaga wnikliwego badania kolejnych wyrazów. Gdy chcemy obliczyć wielkość S, będącą sumą szeregu

$$S = \sum_{i=0}^{\infty} a_i,$$

przybliżamy ją sumą skończoną $S_n = \sum_{i=0}^{n} a_i$, a różnicę $R_n = S - S_n$ nazywamy błędem obcięcia.

Wykonując sumowanie szeregu możemy możemy napotkać różnorakie trudności, np. szereg może być wolnozbieżny, może zawierać oscylujące wyrazy, niektóre wyrazy mogą być wiele rzędów wielkości wieksze niż suma szeregu (małe błędy względne w obliczeniach duży wyrazów dają duże błędy względne wyniku). Takie szeregi nazywane są źle uwarunkowanymi.

Rozważmy szereg naprzemienny postaci

$$\sum_{n=0}^{\infty} b_n \frac{(-x)^n}{n!},$$

gdzie $x \gg 1$ (dla zbyt dużych x może pojawić się przepełnienie) i $b_n > 0$ dla każdego n. Zakładamy, że współczynniki b_n są wolno zmienne w porównaniu z $(-x)^n/n!$, więc stosunek dwóch kolejnych wyrazów może przybliżyć przez

$$-\frac{b_{n+1}}{b_n}\frac{x}{n!} \approx -\frac{x}{n!}.$$

Czyli szereg jest zbieżny dla wszystkich x. Moduły jego wyrazów rosną aż n osiągnie wartość bliską $|x|$. Potem maleją do zera. Liczba wyrazów, które należy uwzględnić przy obliczaniu sumy szeregu, zależy od wymaganej dokładności. Ponieważ jest to szereg naprzemienny, to ostatni wyraz daje dobre oszacowanie błędu obcięcia. Więc wygodnym algorytmem obliczania sumy jest dodawanie kolejnych wyrazów szeregu, dopóki moduł ostatniego stanie się mniejszy niż założona dokładność.

Rozdział 4

Interpolacja i aproksymacja

4.1 Interpolacja

Wiele funkcji jest znanych tylko w ograniczonej liczbie punktów (np. wyniki pomiarów). W takim wypadku interpolacja jest wygodną metodą przybliżania ich wartości w innych punktach. Funkcje interpolujące mogą być potem używane np. do całkowania, różniczkowania oraz rozwiązywania równań nieliniowych i różniczkowych, itd.

4.1.1 Interpolacja wielomianowa

Wielomiany są najprostszymi i najczęściej używanymi funkcjami do interpolacji. Można je bowiem łatwo różniczkować, całkować i obliczać ich wartości. Interpolacja polega na przybliżaniu danej funkcji $f(x)$ przez inną funkcję $g(x)$

$$f(x) = g(x) + E(x),$$

gdzie $E(x)$ jest błędem przybliżenia. Funkcja iterpolująca $g(x)$ powinna pokrywać się z daną funkcją w określonym zbiorze punktów (w węzłach) x_i, $i = 0, 1, \ldots, n$, tzn.

$$f(x_i) = g(x_i), \qquad i = 0, 1, \ldots, n.$$

Następujące twierdzenie gwarantuje istnienie takiego wielomianiu.

Twierdzenie: Jeżeli mamy funkcję rzeczywistą i $n+1$ różnych punktów $x_0, \ldots, x_n$, to istnieje jeden i tylko jeden wielomian o stopniu mniejszym lub równym n, który interpoluje funkcję $f(x)$ w puntach $x_0, \ldots, x_n$.

Rozważmy funkcje

$$l_k(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}, \qquad k = 0, \ldots, n. \tag{4.1}$$

Każda funkcja l_k jest wielomianem stopnia n, więc

$$g_n(x) = \sum_{k=0}^{n} f(x_k) l_k(x) \tag{4.2}$$

jest wielomianem stopnia mniejszego lub równego n. Ponieważ

$$l_k(x_i) = \delta_{ik},$$

to $g_n(x_i) = f(x_i)$ dla $i = 0, \ldots, n$. Wielomian (4.2) nazywany jest wielomianem interpolacyjnym Lagrange'a. Stosuje się również wiele innych formuł interpolacyjnych o nazwach pochodzących od nazwisk najbardziej znanaych matematyków – Newtona, Bessela, Stirlinga, itd.

Można pokazać, że błąd obcięcia dla wielomianiu interpolacyjnego, $E_n(x) = f(x) - g_n(x)$, wyraża się przez

$$E_n(x) = \frac{\prod_{i=0}^{n}(x - x_i)}{(n+1)!} f^{n+1}(\eta), \tag{4.3}$$

gdzie η leży pomiędzy x_0 i x_n. Jeżeli $f(x)$ jest wielomianem stopnia mniejszego lub równego n, to $f^{n+1}(\eta) = 0$ i $E_n(x) = 0$, czyli interpolacja jest dokładna dla takich funkcji.

Mogłoby się wydawać, że dowolną dokładność interpolacji można osiągnąć używając dostatecznie dużej liczby punktów w formule interpolacyjnej. Jednak w praktyce gdy liczba punków interpolacyjnych rośnie błąd początkowo maleje, ale od pewnego momentu zaczyna rosnąć nieograniczenie. Również błędy zaokrągleń rosną wraz z liczbą punktów. A zatem nie powinno stosować się zbyt wielu punktów w formułach interpolacyjnych. Z reguły nie używa się wiecej niż sześciu punktów.

Łatwo zauważyć, że wielomian interpolacyjny Lagrange'a nie jest zbyt przydatny w oblicznach numerycznych. Aby obliczyć jego wartość trzeba wykonać dużo więcej operacji (ok. 2,5 raza) niż jest potrzebnych do obliczenia wartości wielomianu o znanych współczynnikach schematem Hornera. Oprócz tego zwiększenie liczby punktów wiąże się z dodatkowym znacznym wysiłkiem obliczniowym. Z tego powodu w praktycznych zastosowaniach stosowane są inne postacie wielomianu interpolacyjego, np. wzór iterpolacyjny Newtona.

4.2 Interpolacja funkcjami sklejanymi

Rozważmy problem interpolacji danej funkcji zadanej w $n+1$ ztablicowanych punktach ($n \gg 1$). Możemy użyć wielomianu stopnia n przechodzacego przez te punkty, ale jest bardzo prawdopodobne, że taki wielomian będzie „dziko" oscylował między tymi punktami. Inną możliwością jest zbudowanie wielomianu interpolacyjnego w oparciu o kilka sąsiadujących punktów.

W takim wypadku różne wielomiany są przypisane różnym obszarom tabeli punktów. Gdy zmiana wielomianu następuje w węzłach, to funkcja interpolująca jest ciągła, ale w ogólności jej pochodne nie są ciągłe. Jeżeli zbudujemy taką funkcję interpolujacą żadając, by maximalna liczba jej pochodnych była ciągła, to taką funkcje nazywamy funkcją sklejaną (ang. *spline*).

Niech bedą znane wartości funkcji $f(x_i)$, $i = 0, 1, \ldots, n$ i $a = x_0 < x_1 < \ldots < x_n = b$. Szukamy funkcji iterpolującej $g(x)$ na przedziale $\langle a, b\rangle$. Dzielimy ten przedział na n podprzedziałów $P_i = \langle x_{i-1}, x_i\rangle$, $i = 1, \ldots, n$. Najprostszą funkcją $g(x)$ będzie funkcja złożona z liniowych funkcji interpolujących w każdym przedziale P_i. Czyli $f(x)$ jest przybliżana przez łamaną, której pochodna jest w ogólności nieciągła w węzłach. Taka funkcja jest nazywana liniową funkcją sklejaną (ang. *linear spline*). Aby otrzymać bardziej gładką funkcję $g(x)$ możemy użyć wielomianu wyższego stopnia w każdym z podprzedziałów. Najczęsciej stosowany jest wielomian trzeciego stopnia, otrzymujemy wtedy tzw. sześcieną funkcję sklejaną (ang. *cubic spline*).

Sześcienna funkcja interpolująca składa się z wielomianów

$$w_i(x) = a_{3i}(x - x_i)^3 + a_{2i}(x - x_i)^2 + a_{1i}(x - x_i) + a_{0i} \tag{4.4}$$

przypisanych odpowiednio do każdego przedziału P_i. Musimy określić $4n$ współczynników a_{ji}, $j = 0, 1, 2, 3$ i $i = 1, \ldots, n$. W tym celu żądamy, by w_i interpolował $f(x)$ w x_{i-1} i x_i oraz, by jego pierwsza i druga pochodna były ciagłe, tzn.

$$\begin{aligned}
w_i(x_{i-1}) &= f(x_{i-1}), \qquad (i = 1, \ldots, n), && (4.5)\\
w_i(x_i) &= f(x_i), \qquad (i = 1, \ldots, n), && (4.6)\\
w_i'(x_i) &= w_{i+1}'(x_i), \qquad (i = 1, \ldots, n-1), && (4.7)\\
w_i''(x_i) &= w_{i+1}''(x_i), \qquad (i = 1, \ldots, n-1). && (4.8)
\end{aligned}$$

Używając tych warunków możemu otrzymać $4n-2$ równań liniowych na nieznane współczynniki. Brakuje dwóch równań, które otrzymujemy np. przyjmując, że

$$w_1''(x_0) = w_n''(x_v) = 0. \tag{4.9}$$

W ten sposób otrzymujemy tzw. spline naturalny.

4.3 Aproksymacja

W obliczniach numerycznych często musimy przybliżać funkcje. Na przykład gdy chcemy obliczyć pochodną lub całkę funkcji, znanej w postaci tabeli wartości, potrzebujemy przybliżenia tej funkcji, które umożliwia wykonanie tych operacji. Aproksymacja funkcji jest również narzędziem analizy numerycznej, np. rozwiązania równań różniczkowych mogą być przybliżane przez kombinacje prostych funkcji.

Interpolacja, opisana w poprzednich sekcjach, jest oczywiście rodzajem aproksymacji, ale np. gdy funkcja jest zdefiniowna przez dane otrzymane eksperymentalnie, to wartości tej funkcji są obarczone błędem i nie ma potrzeby, by funkcja aproksymująca zgadzała się tymi wartościami. W tej sekcji będziemy rozważać inną metodę przybliżania, która polega na minimalizowaniu wybranej normy błędu przybliżenia.

W szczególności zajmiemy się problemem reprezentacji danych, tzn. będziemy szukać formuły funkcji reprezentujacej duży zbiór niezbyt dokładnych danych $\{(x_i, f(x_i)), i = 1, n\}$, np. pochodzących z pomiaru jakiejś wielkości fizycznej. W takiej sytuacji:

- wartości funkcji obarczone są małą niepewnością lub błędem;
- nie potrzebna jest zbyt duża dokładność i błąd danych wejściowych jest mniejszy od tej dokładności;
- dysponujemy jakąś informacją na temat tej funkcji, np. może istnieć teoria określająca jej postać jako $y = F(x; a, b, c, \ldots)$, gdzie $a, b, c, \ldots$. są parametrami, których najlepszych wartości szukamy.

Szukane przybliżnie jest określone za pomocą pewnej normy błędu, obliczonej dla dla dyskretnego zbioru punktów. Na przykład, jeżeli szukamy funkcji postaci $y = F(x; a, b, c, \ldots)$, to najwygodniejsza norma błędu (norma L_p) jest zdefiniowana następująco

$$\|f - F\|_p = \left(\sum_{i=1}^{n} |f(x_i) - F(x; a, b, c, \ldots)|^p \right)^{1/p}, \tag{4.10}$$

gdzie $i = 1, \ldots, n$. Najczęściej używane wartości p to 1, 2 i ∞. Gdy stosujemy normę

$$\|f - F\|_\infty = \max_{1 \leqslant i \leqslant n} \left(|f(x_i) - F(x; a, b, c, \ldots)| \right), \tag{4.11}$$

otrzmymujemy tzw. przybliżenie minimax, tzn. minimalizowany jest błąd maksymalny. Jednak takie przybliżenie nie jest często stosowane, bo trudno tę normę zminimalizować, gdyż nie jest ona różniczkowalną funkcją parametrów $a, b, c, \ldots$. Ponadto wzmacnia ono rolę pojedynczego dużego błędu.

Najczęściej używa się normy L_2, a otrzymane za jej pomocą przybliżenie nazywane jest aproksymacją najmniejszych kwadratów. Jest to najprostsza norma dająca różniczkowalną i łatwo minimalizowalną funkcję parametrów. Przybliżenie najmniejszych kwadratów nie musi być najlepszym wyborem, ale na pewno jest najławiejsze do otrzymania i analizy.

Aby otrzymać dobre przybliżenie, oprócz normy musimy również wybrać odpowiedni model dla funkcji $F(x; a, b, c, \ldots)$. Właściwy wybór modelu jest ważniejszy niż wybór normy. Zależy on tylko od aproksymowanej funkcji.

Jeżeli jest ona gładka i nie ma wielu ekstremów, to użycie wielomianu niskiego stopnia jest uzasadnione. Wielomiany wysokiego stopnia zazwyczaj nie są przydatne, chyba że mamy powody by wierzyć, że aproksymowana funkcja jest takim wielomianem. Jeżeli funkcja oscyluje, to funkcje trygonometryczne będą najlepsze. Wiekszość aproksymacji jest przeprowadzana za pomocą funkcji liniowych względem parametrów a, b, c, ..., bo takie funkcje są najłatwiejsze w użyciu. Należy podkreślić, że chodzi o liniowość względem parametrów, a nie względem zmiennej x. W takim wypadku funkcję aproksymującą można zapisać w postaci

$$F(x; a, b, c, \ldots) = a\phi_1(x) + b\phi_2(x) + c\phi_3(x) + \ldots, \tag{4.12}$$

gdzie $\{\phi_i(x), i = 1, 2, \ldots, m\}$ jest zbiorem odpowiednio wybranych funkcji. Takie przybliżenie jest nazywane liniowym, mimo że funkcje $\phi_i(x)$ mogą być funkcjami nieliniowymi. Na przykład, jeżeli weźmiemy $\phi_i(x) = x^{i-1}$, to funkcja $F(x; a, b, c, \ldots)$ będzie wielomianem stopnia co najwyżej $m - 1$. Kładąc $\phi_i(x) = \sin ix$ otrzymamy aproksymację Fouriera. Użycie liniowego przybliżenia prowadzi do równań liniowych określających parametry a, b, c, ..., które można z łatwością rozwiązać.

W wielu przypadkach zamiana zmiennych może poprawić znacząco przybliżenie. Na przykład, gdy aproksymowana funkcja zmienia się w zakresie kilku rzędów wielkości, to może się okazać, że warto ją zlogarytmować przed wykonaniem aproksymacji wielomianowej. Pewne problemy mogą wymagać użycia nieliniowch funkcji parametrów, a co za tym idzie rozwiązywania równań nieliniowych, co z z reguły jest dużo trudniejsze. Z tego powodu zazwyczaj unika się stosowania takich modeli, mimo że mogą dać dużo lepsze wyniki.

Analiza błędu aproksymacji jest bardzo trudna. W ogólności możemy powiedzieć, że błąd, określony jako norma sumy różnic między wartościami funkcji aproksymowanej i aproksymującej w tych samych węzłach, maleje wraz ze wzrostem liczby funkcji bazowych ϕ_i. Jednak nic nie możemy powiedzieć o błędzie aproksymacji względem funkcji, którą przybliżamy, bo jej po prostu nie znamy.

4.4 Liniowa metoda najmniejszych kwadratów

Niech $f(x)$ będzie funkcją, której wartości znamy w puktach $\{x_1, x_2, \ldots, x_n\}$, a $\{\phi_1, \phi_2, \ldots, \phi_n\}$ zbiorem funkcji bazowych, za pomocą których będziemy szukać liniowego przybliżenia

$$f(x) \approx F(x, \vec{a}) = \sum_{j=1}^{m} a_j^{(m)} \phi_j. \tag{4.13}$$

Indeks górny m oznacza, że współczynniki $a_j^{(m)}$ w ogólności zależą od liczby użytych funkcji bazowych. Otrzymujemy je minimalizując kwadrat normy

L_2 równy

$$\alpha^2 = \sum_{i=1}^{n} \left(f(x_i) - \sum_{j=1}^{m} a_j^{(m)} \phi_j(x_i) \right)^2, \tag{4.14}$$

którego minimum otrzymujemy rozwiązując układ równań

$$\frac{\partial \alpha^2}{\partial a_k^{(m)}} = -2 \sum_{i=1}^{n} \left(f(x_i) - \sum_{j=1}^{m} a_j^{(m)} \phi_j(x_i) \right) \phi_k(x_i), \tag{4.15}$$

gdzie $k = 1, \ldots, m$. Ten układ m równań liniowych z m niewiadomymi nazywany jest układem normalnym. Macierz współczynników będzie nieosobliwa, o ile wszystkie punkty x_i będą różne i funkcje bazowe liniowo niezależne. Jeżli $m \geqslant n$, tzn. gdy liczba funkcji bazowych jest niemniejsza niż liczba węzłów, to w ogólności można znaleźć rozwiązanie dające $\alpha^2 = 0$, czyli otrzymana funkcja aproksymująca będzie interpolować funkcję $f(x)$. Układ normalny jest zazwyczaj źle uwarunkowany dla $m \gg 1$.

Jeżli wartości $f(x_i)$, $i = 1, \ldots, n$, wstawimy kolejno do równania (4.13), to otrzymamy nadokreślony układ n równań z m niewiadomymi postaci $A\vec{a} = \vec{b}$, gdzie A jest macierzą $n \times m$, $n \geqslant m$, a $\vec{b} = (f(x_1), f(x_2), \ldots, f(x_n))$. Zdefiniujmy wektor reszt

$$\vec{r}(\vec{a}) = \vec{b} - A\vec{a}. \tag{4.16}$$

Wtedy rozwiązaniem układu nadokreślonego nazywamy wektor $\vec{a}^*$ minimalizujący normę L_2 wektora reszt, tzn.

$$\|\vec{r}(\vec{a}^*)\|_2 = \min_{\vec{a}} \|\vec{r}(\vec{a})\|_2 = \min_{\vec{a}} \left(\sum_{i=1}^{n} |r_i(\vec{a})|^2 \right)^2. \tag{4.17}$$

Wektor $\vec{a}^*$ nazywamy rozwiązaniem średniokwadratowym, bo otrzymuje się go za pomocą normy L_2. W praktyce używane są również inne normy.

Wektor $\vec{a}^*$ spełnia również tzw. układ normalny postaci

$$A^T A\vec{a} = A^T \vec{b}. \tag{4.18}$$

Aby to pokazać załóżmy, że $\vec{a}$ jest dowolnym wektorem n-elementowym. Do wektora reszt (4.16) dodajmy i odejmijmy wektor $A\vec{a}^*$, wtedy

$$\vec{r} = \vec{b} - A\vec{a}^* + A\vec{a}^* - A\vec{a}^* = \vec{r}^* + A(\vec{a}^* - \vec{a}). \tag{4.19}$$

Obliczmy kwadrat normy L_2 wektora $\vec{r}$

$$\|\vec{r}\|_2^2 = \vec{r}^T \vec{r}. \tag{4.20}$$

Korzystając z (4.19) dostajemy

$$\begin{aligned} \|\vec{r}\|_2^2 &= \|\vec{r}^*\|_2^2 + \\ &\quad [A(\vec{a}^* - \vec{a})]^T \vec{r}^* + \vec{r}^{*T} A(\vec{a}^* - \vec{a}) + \\ &\quad \|A(\vec{a}^* - \vec{a})\|_2^2. \end{aligned} \tag{4.21}$$

Drugi wyraz (liczbę) po prawej stronie (4.21) można zgodnie z regułami transponowania macierzy zapisać jako $2(\vec{a}^* - \vec{a})^T A^T \vec{r}^*$. Ponieważ $\vec{r}^*$ jest rozwiązaniem (4.18), to $A^T \vec{r}^* = 0$ i

$$\|\vec{r}\|_2^2 = \|\vec{r}^*\|_2^2 + \|A(\vec{a}^* - \vec{a})\|_2^2. \tag{4.22}$$

Stąd dostajemy oszacowanie $\|\vec{r}\|_2^2 \geqslant \|\vec{r}^*\|_2^2$, czyli wektor $\vec{a}^*$ minimalizuje normę L_2 wektora reszt.

4.5 Wielomiany w MATLAB-ie

W MATLAB-ie wielomian jest reprezetowany przez wektor jego współczynników zapisanych w porządku malejącym, tzn. pierwszym elementem wektora jest współczynnik wyrazu wielomianu o najwyższej potędze zmiennej niezależnej.

MATLAB dostarcza wielu funkcji pozwalających na różnorakie operacje na wielomianach. Należą do nich:

conv(a, b) – obliczająca wektor wspólczynników wielomianu będącego iloczynem wielomianów, reprezentowanych przez wektory a i b;

$[i, r] =$ *deconv*(a, b) – zwracająca wektory i i r, reprezentujące, odpowiednio, iloraz wielomianów danych przez wektory a i b oraz resztą z tego dzielenia;

roots(a) – oblicza pierwiastki wielomianu, reprezentowanego przez wektor a;

poly(p) – zwraca wektor wspólczynników wielomanu, którego pierwiastkmi są elementy wektora p;

polyval(a, x) – oblicza wartości wielomianu, reprezentowanego przez wektor a, w punktach danych liczbą, wektorem lub macierzą x;

polyder(a) – zwraca wektor współczynników wielomianu, będącego pochodną wielomianu danego wektorem a;

polyint(a, k) – zwraca wektor współczynników wielomianu, będącego całką wielomianu danego wektorem a, k jest wartością stałej całkowania.

4.6 Interpolacja i aproksymacja w MATLAB-ie

Problem aproksymacji i interpolacji wielomianowej może być rozwiązany w MATLAB-ie za pomocą funkcji *polyfit*. Załóżmy, że n-elementowe wektory x i y zawierają, odpowiednio, węzły i wartości aproksymowanej funkcji. Polecenie

$$p = polyfit(x, y, k) \tag{4.23}$$

zwróci wektor p, reprezentujący wielomian stopnia k, minimalizujacy normę L_2 błedów. Jezeli $k \geqslant n$, to wygenerowany wielomian będzie wielomianem interpolacyjnym. Na przykład następujący ciąg poleceń zbuduje wektor węzłów, wektor losowych wartości funkcji i wielomian aproksymacyjny 3-go stopnia

$$\begin{aligned} x &= 0:10 \\ y &= rand(1, length(x)) \\ p &= polyfit(x, y, 3) \end{aligned}$$

Funkcja $rand()$ generuje liczbę pseudolosową o rozkładzie jednorodnym z przedziału (0,1), $rand(n)$ macierz $(n \times n)$ takich liczb, $rand(n, m)$ macierz $(n \times m)$ takich liczb, itd. Aby zilustrować otrzymane wyniki, możemy skorzystać z funkcji *plot*, która w okienku graficznym wyświetla wykresy dwuwymiarowe. Polecania

$$\begin{aligned} xx &= 0:0.1:10 \\ yy &= polyval(p, xx) \\ &plot(x, y,' o', xx, yy,' -') \end{aligned}$$

zwrócą gęstszy wektor xx węzłów na rozważanym przedziale, wektor yy wartości wielomianu aproksymującego p w węzłach xx, a w okienku graficznym pojawią się punkty, oznaczone symbolem o, o współrzędnych danych przez odpowiednie elementy wektorów x i y oraz linia ciągła (opcja $'-'$), będąca wykresem wielomianu aproksymującego, określonego przez wektory xx i yy.

Jeżeli przez te same punkty chcielibyśmy przeprowadzić sześcienną funkcje sklejaną (ang. *cubic spline*), to możemy użyć funkcji *spline*

$$yy = spline(x, y, xx)$$

W MATLAB-ie jest dostępne również funkcja *interp*, umożliwiająca interpolację funkcjami zdefinowanymi przedziałami (niekoniecznie sklejanymi) o analogicznej składni

$$yy = interp(x, y, xx,' metoda'),$$

gdzie opcja *metoda* definuje rodzaj interpolanta:

nearest – funkcja najbliższego sąsiedztwa, tzn. interpolant składa się z poziomych odcinków, których położenia i długości są określone przez wartości aproksymowanej funkcji oraz węzły;

linear – interpolacja liniowa (łamaną);

spline – interpolacja sześcienną funkcją sklejaną (w zasadze to samo co funkcja *spline*)

pchip **lub** *cubic* – przedziałami zdefinowany sześcienny interpolant Hermite'a; na krańcach każdego przedziału interpolant ma nie tylko określoną wartość, ale również wartość pierwszej pochodnej, tak dobraną, by zachowywał kształt aproksymowanej funkcji i jej monotoniczność (w odróżnieniu od sklejanej funkcji kubicznej, nie musi on mieć ciągłej drugiej pochodnej).

Liniową aproksymację najmniejszych kwadratów może przeprowadzić w MATLAB-ie budując odpowiedni układ nadokreślony i rozwiązując go bezpośrednio za pomocą operacji lewego dzielenia \, które wypadku układów nadokreślonych daje rozwiązanie średniokwadratowe lub rozwiązując układ normalny (4.18).

Rozdział 5

Różniczkowanie i całkowanie numeryczne

5.1 Numeryczna aproksymacja pochodnej

Funkcja $f(x)$, która ma być zróżniczkowana, może być zadana wzorem lub w postaci dyskretnych punktów. W ogólności te pierwsze funkcje mogą być zróżniczkowane dokładnie. Różniczkowanie dyskretnych danych wymaga przybliżającej procedury numerycznej. Formuły różniczkowania numerycznego mogą zostać wyprowadzone przez dopasowanie funkcji aproksymującej (np. wielomianu) do dyskretnych danych i zróżniczkowanie tej funkcji. Jednak jeżeli przeprowadzimy przez zadane punkty wielomian interpolacyjny, to jego pochodna nie musi dawać bardzo dokładnego przybliżenia pochodnej $f'(x)$ nawet w węzłach. W ogólności numeryczne różniczkowanie jest bardzo niedokładne.

W praktyce używa się wielu metod rożniczkowania numerycznego: można zróżniczkować interpolujący dane wielomian w postaci ogólnej lub Lagrange'a. Gdy węzły sa równoodległe stosuje sie wielomiany Newtona (progresywne i wsteczne). Formuły różniczkowania numerycznego można wyprowadzić również za pomocą rozwinięcia Taylora. W takim wypadku dyskretyzujemy dziedzinę rożniczkowanej funkcji, zastępując ją siatką równoodległych punktów x_i ($x_{i+1} - x_i = h = const$). Wartość funkcji w węźle oznaczamy przez $f_i = f(x_i)$. Zacznijmy od rozwnięcia Taylora dla $f_{i\pm1} = f(x_i \pm h)$

$$f(x_i + h) = f(x_i) + f'(x_i) + \frac{1}{2!}f''(x_i)h^2 + \frac{1}{3!}f'''(\eta_1)h^3, \qquad (5.1)$$

$$f(x_i - h) = f(x_i) - f'(x_i) + \frac{1}{2!}f''(x_i)h^2 - \frac{1}{3!}f'''(\eta_2)h^3. \qquad (5.2)$$

Odejmując od (5.1) równość (5.2) otrzymujemy

$$f(x_i + h) - f(x_i - h) = 2f'(x_i)h + \frac{1}{3!}\left[f'''(\eta_1) + f'''(\eta_2)\right]h^3. \quad (5.3)$$

Jeżeli $f'''(x)$ jest ciągła, to istnieje takie η, że $\left[f'''(\eta_1) + f'''(\eta_2)\right]/2 = f'''(\eta)$. Czyli

$$f'(x_i) = \frac{f(x_i + h) - f(x_i - h)}{2h} - \frac{1}{3!} f'''(\eta) h^2 \tag{5.4}$$

lub

$$f'_i = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2), \tag{5.5}$$

gdzie $f'_i = f'(x_i)$. W ten sposób otrzymaliśmy przybliżenie pochodnej przez centralny (symetryczny) wzór różnicowy. Jest on również nazywany wzorem trójpunktowym, bo można go otrzymać różniczkując wielomian interpolacyjny rozpięty na punktach (x_{i-1}, f_{i-1}), (x_i, f_i) i (x_{i+1}, f_{i+1}), ale do jego obliczenia potrzebne są tylko dwie wartości funkcji.

Stosując analogicznę technikę, tzn. odpowiednio kombinując rozwinięcia Taylora w różnych punktach siatki, możemy otrzymać centralne przyliżenia różnicowe pochodnych różnych rzędów i o różnych dokladnościach, np.

$$f'_i = \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12h} + O(h^4), \tag{5.6}$$

$$f''_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2), \tag{5.7}$$

$$f''_i = \frac{-f_{i+2} + 16f_{i+1} + 30f_i + 16f_{i-1} - f_{i-2}}{12h^2} + O(h^4), \tag{5.8}$$

gdzie $f''_i = f''(x_i)$. Jeżeli mamy do dyspozycji wartości funkcji w węzłach leżących tylko po jednej stronie węzła x_i, to centralne formuły różnicowe nie mogą być użyte. Wyprowadza się wtedy formuły progresywne (wsteczne), dla węzłów położonych po prawej (lewej) stronie węzła, np.

$$f'_i = \frac{-f_i + f_{i+1}}{h} + O(h), \tag{5.9}$$

$$f'_i = \frac{f_i - f_{i-1}}{h} + O(h), \tag{5.10}$$

$$f'_i = \frac{-3f_i + 4f_{i+1} - f_{i+2}}{2h} + O(h^2), \tag{5.11}$$

$$f'_i = \frac{3f_i - 4f_{i-1} + f_{i-2}}{2h} + O(h^2), \tag{5.12}$$

$$f''_i = \frac{f_i - 2f_{i+1} + f_{i+2}}{h^2} + O(h), \tag{5.13}$$

$$f''_i = \frac{f_i - 2f_{i-1} + f_{i-2}}{h^2} + O(h), \tag{5.14}$$

$$f''_i = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{h^2} + O(h^2), \tag{5.15}$$

$$f''_i = \frac{2f_i - 5f_{i+1} + 4f_{i+2} - f_{i+3}}{h^2} + O(h^2). \tag{5.16}$$

W ogólności formuły centralne dają, przy tej samej liczbie użytych wartości funkcji, dokładniejsze wyniki niż formuły progresywne lub wsteczne.

Bardzo ważnym czynnikiem wpływającym na dokładność numerycznego obliczania pochodnych jest błąd zaokrąglenia. Zbadamy jego rolę na przykładzie formuły (5.5). Załóżmy, że dokładne wartości funkcji $f_{i\pm1}$ zostały przez komputer zaokrąglone do $g_{i\pm1}$, a odpowiednie błędy zaokrąglenia są równe $\epsilon_{i\pm1}$. Czyli przybliżenie pochodnej jest obarczone błędem ϵ będącym sumą błędów zaokrąglenia ϵ_z i obcięcia ϵ_o

$$f'_i = \frac{g_{i+1} - g_{i-1}}{2h} + \epsilon, \tag{5.17}$$

gdzie (por. (5.4))

$$\epsilon = \epsilon_z + \epsilon_o = \frac{\epsilon_1 - \epsilon_{-1}}{2h} - \frac{1}{6} f'''(\eta) h^2. \tag{5.18}$$

Jeżeli $|\epsilon_1| \leqslant \epsilon_m$, $|\epsilon_{-1}| \leqslant \epsilon_m$ i $M = \max_x[|f'''(x)|]$, to

$$\epsilon \leqslant \frac{\epsilon_m}{h} + \frac{M}{6} h^2. \tag{5.19}$$

Wartością h, która minimalizuje prawą stronę tej nierówności jest $h = (3\epsilon_m/M)^{1/3}$. Gdy h jest względnie duże, błąd zaokrąglenia jest mały. Dominuje błąd obcięcia, który maleje z malejącym h. Jednak gdy h jest relatywnie małe zaczyna dominować błąd zaokrąglenia, który rośnie wraz z maleniem h.

5.2 Całkowanie numeryczne

5.2.1 Kwadratury Newtona-Cotesa

Całkowanie numeryczne jest podstawowym narzędziem umożliwiający obliczenie przybliżonej wartości całki oznaczonej

$$I[f] = \int_a^b f(x)dx, \tag{5.20}$$

ktora nie może być obliczona analitycznie. Załóżmy, że znamy wartości funkcji $f(x)$ w węzłach $a = x_0 < x_1 < \ldots < x_n = b$, wtedy suma

$$Q[f] = \sum_{i=0}^{n} w_i f(x_i), \tag{5.21}$$

o następującej własności

$$I[f] = Q[f] + R, \tag{5.22}$$

nazywana jest kwadraturą (wagi w_i są stałymi, R jest błędem kwadratury). Standardową metodą znajdowania kwadratur jest zastępowanie funkcji podcałkowej wielomianem interpolacyjnym i następnie jego scałkowanie. Zinterpolujmy funkcję $f(x)$ używając wielomianów interpolacyjnych Lagrange'a (por. (4.2) i (4.3))

$$f(x) = \sum_{i=0}^{n} f(x_i) l_i(x) + E_n(x). \tag{5.23}$$

Całkując tę funkcję otrzymujemy

$$I[f] = \sum_{i=0}^{n} f(x_i) \int_a^b l_i(x)dx + \int_a^b E_n(x) = \sum_{i=0}^{n} w_i f(x_i) + E_n. \tag{5.24}$$

Tak otrzymane kwadratury w wypadku węzłów równoodległych ($x_{i+1} - x_i = h$) nazywane są kwadraturami Newtona-Cotesa.

Jeżeli $n = 1$, to z (4.1) mamy $l_0 = (x - x_1)/(x_0 - x_1) = -(x - b)/h$ i $l_1 = (x - x_0)/(x_1 - x_0) = (x - a)/h$. Stąd otrzymujemy

$$w_0 = \int_a^b l_0(x)dx = -\frac{1}{h}\int_a^b (x - b)dx = \frac{1}{2h}(a - b)^2 = \frac{h}{2}. \tag{5.25}$$

W analogiczny sposób dostajemy $w_1 = h/2$. Błąd przybliżenia jest określony przez

$$\begin{aligned} E_1 &= \int_a^b E_1(x)dx = \frac{f''(\eta)}{2!}\int_a^b (x - x_0)(x - x_1)dx \\ &= \frac{f''(\eta)}{2}\int_a^b (x - a)(x - b)dx = -\frac{1}{12}(b - a)^3 f''(\eta) \\ &= -\frac{h^3}{12} f''(\eta). \end{aligned} \tag{5.26}$$

W ten sposób otrzymaliśmy kwadratrę trapezów (wstawiamy za $a = x_0$ i $b = x_1$)

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}(f_0 + f_1) - \frac{h^3}{12} f''(\eta), \tag{5.27}$$

gdzie $f_i = f(x_i)$. Kładąc kolejno $n = 2, 3$ i całkując odpowiednie wielomiany $l_i(x)$ oraz reszty $E_n(x)$ może wyprowadzić kolejne kwadratury:

Simpsona

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) - \frac{h^5}{90} f'''(\eta); \tag{5.28}$$

3/8 Simpsona

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3) - \frac{3h^5}{80} f^{(4)}(\eta).$$

Aby porównać „uczciwie" rożne kwadratury powinniśmy używać tej samej liczby wartości funkcji f_i. Np. jeżeli mamy trzy wartości f_0, f_1 i f_2, to całkę $\int_{x_0}^{x_2} f(x)dx$ możemy obliczyć używając dwóch kwadratur trapezów

$$\begin{aligned}\int_{x_0}^{x_2} f(x)dx &= \int_{x_0}^{x_1} f(x)dx + \int_{x_1}^{x_2} f(x)dx \\ &\approx \frac{h}{2}(f_0+f_1)+\frac{h}{2}(f_1+f_2)=\frac{h}{2}(f_0+2f_1+f_2) \quad (5.29)\end{aligned}$$

Tę całkę można przybliżyć, używając tych samych wartości funkcji, kwadraturą Simpsona (5.28). Wzór (5.29) jest przykładem kwadratury złożonej. Możemy go ugólnić zakładając, że przedział $[a,b]$ został podzielony na M podprzedziałów $[x_k, x_{k+1}]$ o długości $h=(b-a)/M$, za pomocą równoodległych węzłów $x_k = a+kh$, $k=0,1,\ldots,M$

$$\begin{aligned}\int_a^b f(x)dx &= h\left(\frac{f_0}{2}+f_1+\ldots+f_{M-1}+\frac{f_M}{2}\right) \\ &\quad -\frac{(b-a)h^2 f''(\eta)}{12}. \qquad (5.30)\end{aligned}$$

Z analizy błędu obcięcia wynika, że jeżeli występująca w nim druga pochodna jest ograniczona, to możemy uzyskać dowolną dokładność biorąc dostatecznie dużą wartość M. W praktyce kładziemy $M=1$ i a potem w każdym następnym kroku dzielimy każdy przedział na dwa podprzedziały. Jest to wygodne, bo możemy korzystać z poprzednio otrzymanych wartości funkcji. Musimy tylko obliczyć jej wartości w nowych punktach podziału. Takie dzielenie kończymy gdy osiągnięta zostanie satysfakcjonująca dokładność lub gdy wynik już się nie zmienia.

Złożona kwadratura Simpsona ma postać

$$\begin{aligned}\int_a^b f(x)dx &= \frac{h}{3}\left(f_0+f_M+4\sum_{i=1}^{M/2} f_{2i-1}+2\sum_{i=1}^{M/2-1} f_{2i}\right) \\ &\quad -\frac{(b-a)h^4 f^{(4)}(\eta)}{180} \qquad (5.31) \\ &= \frac{h}{3}\left(S_{0M}+4S_{nieparz}+2S_{parz}\right)-\frac{(b-a)h^4 f^{(4)}(\eta)}{180},\end{aligned}$$

gdzie założono, że liczba podprzedziałów, do których zastosowano proste kwadratury Simpsona jest równa $M/2$, czyli M jest liczbą parzystą.

5.2.2 Kwadratury Gaussa

Rozważmy kwadraturę (5.21), ale teraz nie będzimy zakładać, że węzły są równoodległe, czyli kwadratura będzie określona przez $2n+2$ niezależnych stałych x_i i w_i. Znajdujemy je zakładając, że kwadratura jest dokładna dla

wielomianów stopnia niewyższego niż $2n+1$. Dla $n=1$, oznacza to, że błąd przybliżenia całki

$$\int_{-1}^{1} f(x)dx = w_0 f(x_0) + w_1 f(x_1) + E_1 \tag{5.32}$$

będzie równy 0 dla $f(x)=1$, x, x^2 i x^3 (dla wygody przjęliśmy $[-1,1]$ jako przedział całkowania). Wstawiając kolejno te funkcje do (5.32) dostajemy układ równań nieliniowych

$$\begin{aligned} w_0 + w_1 &= 2, && (5.33)\\ x_0 w_0 + x_1 w_1 &= 0, && (5.34)\\ x_0^2 w_0 + x_1^2 w_1 &= \frac{2}{3}, && (5.35)\\ x_0^3 w_0 + x_1^3 w_1 &= 0, && (5.36)\\ & && (5.37) \end{aligned}$$

którego rozwiązaniem jest $x_0 = -x_1$ i $w_0 = w_1$. Podobnie możemy wyprowadzić kwadratury zawierające więcej punktów, ale rozwiązanie otrzymanych układów równań nieliniowych może być bardzo trudne. Tak zbudowane kwadratury nazywane sa kwadraturami Gaussa-Legendre'a.

Jeżeli chcemy za pomocą kwadratury Gaussa-Legendre'a obliczyć całką na przedziale $[a,b]$, to możemy ją sprowadzić do przedziału [-1,1] używając transformacji

$$x = \frac{b-a}{2}t + \frac{b+a}{2} \tag{5.38}$$

by otrzymać

$$\begin{aligned} \int_a^b f(x)dx &= \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right)dt \\ &\approx \frac{b-a}{2}\sum_{i=0}^{n} w_i f\left(\frac{b-a}{2}t_i + \frac{b+a}{2}\right). \end{aligned} \tag{5.39}$$

Podobnie jak w wypadku kwadratur Newtona-Cotesa, możemy rozważać złożone formuły Gaussa-Legendre'a, dzieląc przedział całkowania na podprzedziały i stosując w każdym z nich kwadraturę prostą.

Kwadratury Gaussa są bardzo użyteczne, gdy chcemy obliczyć całki postaci

$$\int_a^b w(x)f(x)dx, \tag{5.40}$$

gdzie funkcja $w(x)$ jest nazywana funkcją wagową. Dla różnych funkcji wagowych i przedziałów całkowania otrzymujemy różne zbiory węzłów i wag

kwadratur, np. gdy $w(x) = 1$ i $[a, b] = [-1, 1]$, to stosujemy, opisane wyżej, kwadratury Gaussa-Legendra'a, gdy $w(x) = 1/\sqrt{1-x^2}$ i $[a, b] = [-1, 1]$, kwadratury Gaussa-Czebyszewa, gdy $w(x) = e^{-x}$ i $[a, b] = [0, \infty]$, kwadratury Laguerre'a oraz gdy $w(x) = e^{-x^2}$ i $[a, b] = [-\infty, \infty]$ kwadratury Hermite'a.

5.2.3 Całkowanie w MATLAB-ie

Przybliżoną wartość całki jednokrotnej można otrzymać w MATLAB-ie na wiele sposobów. Dla stabelaryzowanch funkcji podcałkowych przeznaczona jest funkcja *trapz*

$$trapz(x, y), \tag{5.41}$$

gdzie x jest wektorem węzłów, y wektorem wartości funkcji podcałkowej w tych węzłach. Jak sie można domyślać zaimplementowano metodę trapezów, ale węzły nie muszą być równoodległe.

Inne funkcje, np. *quad*, wymagają znajomości wzoru całkowanej funkcji. Aby obliczyć całkę $\int_a^b f(x)dx$ wprowadzamy polecenie

$$quad(fun, a, b) \tag{5.42}$$

gdzie fun jest uchwytem do zakodowanej w MATLAB-ie funkcji $f(x)$. Funkcja ta powinnna przyjmować wektorowy argument x (tam gdzie jest to konieczne należy używać operacji kropkowych). Granice całkowania muszą być skończone. Funkcja *quad* używa kwadratur Simpsona.

Do dyspozyji mamy też inne funkcje o analogicznej składni:

quadl – stosuje kwadratury Loabtto, a i b muszą być skończone;

quadgk – używa kwadratury Gaussa-Kronoda, a i b mogą Inf i $-Inf$, gdzie Inf ($-Inf$) zwraca maszynową reprezentację plus (minus) nieskończoności (jest np. wynikiem dzielnia przez zero lub przepełnienia).

Rozdział 6

Rozwiązywanie równań różniczkowych

Równania różniczkowe są najważniejszym narzędziem matematycznym opisującym różne zjawiska fizyczne. Jeżeli równanie różniczkowe ma tylko jedną zmienną niezależną, to nazywane jest równaniem różniczkowym zwyczajnym. Gdy jest ich więcej, to mamy do czynienia z równaniem różniczkowym cząstkowym. Większość równań różniczkowych, mających zastosowanie w fizyce, to równania cząstkowe. Jednak są one z reguły trudne do rozwiązania, więc często upraszcza się problem redukując równania cząstkowe do zwyczajnych.

W tym rozdziale zajmiemy się rozwiązywaniem numerycznym zagadnienia początkowego dla równań i układów równań różniczkowych zwyczajnych, tzn. warunki brzegowe nałożone na te równania bedą określone w jednym punkcie. Problemy, wymagające rozwiązywania równań różniczkowych, mogą zazwyczaj być zredukowane do rozwiązywania układów równań rożniczkowych pierwszego rzędu, np. równanie n-tego rzędu

$$\frac{d^n y}{dt^n} = f\left(t, y, \frac{dy}{dt}, \ldots, \frac{dy^{n-1}}{dt^{n-1}}\right) \tag{6.1}$$

można sprowadzić do układu n równań pierwszego rzędu za pomocą transformacji $y_1 = y$ i $y_{i+1} = \frac{d^i y}{dt^i}$ dla $j = 1, \ldots, n-1$; otrzymujemy w ten sposób układ

$$\begin{aligned} \frac{dy_i}{dt} &= y_{i+1}, \qquad i = 1, 2, \ldots, n-1, \\ \frac{dy_n}{dt} &= f\left(t, y_1, y_2, \ldots, y_n\right). \end{aligned} \tag{6.2}$$

Z tego powodu będziemy zajmować się rozwiązywaniem układów rownań różniczkowych zwyczajnych pierwszego rzędu, które zapisujemy w postaci

$$\frac{dy_i}{dt} = f_i\left(t, y_1, y_2, \ldots, y_n\right), \qquad i = 1, 2, \ldots, n. \tag{6.3}$$

Większość metod rozwiązywania zagadnienia początkowego dla układów równań różniczkowych jest prostym uogólniem metod dla jednego równania. Zatem będziemy zajmować się rozwiązywaniem równania

$$\frac{dy}{dt} = f(t, y). \tag{6.4}$$

W ogólności numeryczne rozwiązywanie zagadnienia początkowego jest otrzymywane krok po kroku, tzn. używając wartości rozwiązania dla $t \leqslant t_i$, obliczamy tę wartość dla późniejszej chwili $t_{i+1} = t_i + h_i$. Ten proces jest powtarzany dla $i = 0, 1, 2, \ldots$. W pierwszym kroku ($i = 0$) rozwiązanie nie jest znane dla $t < t_0$, więc możemy użyć tylko wartości początkowych w chwili $t = t_0$, by znaleźć rozwiązanie w t_1. Zatem metody, które wykorzystują więcej niż jedną poprzednią wartość, wymagają użycia innych technik, dostarczających potrzebnej liczby wartości startowych. Takie metody nazywane są wielokrokowymi. W najprostszym wypadku używamy informacji tylko z poprzedniego kroku, więc obliczana wartość $y(t_{i+1})$ zależy tylko od $y(t_i)$. Takie metody noszą miano jednokrokowych.

Załóżmy, że długość kolejnych kroków jest stała, tzn. $t_i = t_0 + ih$. Wtedy metodę całkowania numerycznego, wykorzystującą ustaloną liczbę poprzednich wartości zapisujemy w postaci

$$y_{i+1} = \sum_{k=0}^{n} a_k y_{i-k} + h \sum_{k=-1}^{n} b_k y'_{i-k}, \tag{6.5}$$

gdzie $y_j = y(t_j)$ i $y'_j = f(t_j, y_j)$. Ta formuła używa informacji z $n + 1$ poprzednich punktów.

Jeżeli $b_{-1} = 0$, wtedy y_{i+1}, dane przez (6.5), jest liniową kombinacją znanych poprzednich wartości y_{i-k} i ich pochodnych. Taka formulą jest nazywana *jawną*. Przykładem takiej formuły jest metoda Eulera

$$y_{i+1} = y_i + hy'_i. \tag{6.6}$$

Metody jawne ekstrapolują wartość rozwiązania w następnym punkcie używając poprzednich punktów i w ogólności są obarczone dużym błędem.

Gdy $b_{-1} \neq 0$, wtedy y_{i+1} występuje po prawej stronie (6.5), bo $y'_{i+1} = f(t_{i+1}, y_{i+1})$. Taka formuła nosi nazwę *niejawnej*. Aby otrzymać rozwiązanie w następnym punkcie, należy rozwiązać równanie algebraiczne. Jednak ten dodatkowy wysiłek obliczniowy jest kompensowany przez wyższą dokładność i lepszą stabilność tych metod. Przykładem takiej formuły jest metoda trapezów

$$y_{i+1} = y_i + \frac{h}{2}(y'_i + y'_{i+1}). \tag{6.7}$$

Ponieważ formuły całkowania numerycznego przybliżają ciągłe operacje przez dyskretne sumy, są obarczone błędem obcięcia. Jeżeli wartość y_i w poprzednim punkcie jest znana dokładnie, to błąd obcięcia jest równy błędowi

całkowania w ostatnim kroku. Na przykład używając rozwinięcia Taylora otrzymujemy błąd obcięcia dla metody Eulera równy $\frac{1}{2}h^2y''(\eta)$. Taki błąd nazywany jest lokalnym błędem obcięcia. W ogólności można oczekiwać, że lokalny błąd obcięcia będzie postaci $ch^{r+1}y^{(r+1)}(\eta)$, gdzie r jest rzędem formuły.

6.1 Stabilność numerycznych metod całkowania

Większość metod numerycznych rozwiązywania zagadnień początkowych polega na wykonywaniu kolejnych kroków, tzn. wartość rozwiązania w następnym punkcie zależy od jego wartości w poprzednich punktach. Zatem są dwa źródła błędów obcięcia wartości y_{n+1} – błąd spowodowany przez n-ty krok i błąd wartości y_n. Pierwszy z nich nazywany jest lokalnym błędem obcięcia, drugi jest wynikiem propagacji tego błędu obcięcia. Pierwszy z nich jest względnie łatwy do oszacowania. Drugi, trudny do określenia, może powodować problemy obliczeniowe, które opisuje się za pomocą pojęcia stabilności numerycznej metody całkowania. Jeżeli mały błąd, pojawiający się na pewnym etapie całkowania, szybko rośnie w kolejnych krokach, to metoda jest nazywana niestabilną, jeżeli pozostaje on w jakimś sensie ograniczony, to jest ona stabilna.

Ponieważ błąd obcięcia zależy od długości kroku, to stabilność metody też jest z nim związana. Dla za dużych kroków większość metod da bezsensowne wyniki, więc warto badać stabilność tylko dla sensownie małych. Jeżeli dla każdego równania różniczkowego istnieje wartość kroku h_0, dla której zmiana wartości początkowej o ustaloną wielkość daje ograniczoną zamianę rozwiązania numerycznego dla wszystkich $h \in [0, h_0]$, to mówimy, że metoda jest stabilna. Stabilność metody całkowania numerycznego oznacza, że rozwiązanie nie jest wrażliwe na zaburzenia, ale nie musi być ono dokładne – dokładność jest związana z pojęciem zbieżności. Metoda jest zbieżna, jeżeli rozwiązanie o dowolnej dokładności można otrzymać wybierając dostatecznie mały krok. Definiując zbieżność zakłada się brak błędów zaokrąglenia, ale w praktyce błędy zaokrąglenia rosną wraz z maleniem kroku i osiągnięcie dowolnie dużej dokładności nie jest możliwe.

Ponieważ jest bardzo trudno analizować stabilność metody całkowania numerycznego dla dowolnego rownania nieliniowego, zazwyczaj rozważa się proste liniowe równanie różniczkowe postaci

$$\frac{dy}{dt} = \lambda y, \qquad y(0) = y_0, \tag{6.8}$$

którego rozwiązaniem jest $y = y_0 e^{\lambda t}$. Dla tego równania metoda całkowania numerycznego (6.5) daje

$$(1 - h\lambda b_{-1})y_{i+1} = \sum_{k=0}^{m} (a_k + h\lambda b_k) y_{i-k}. \tag{6.9}$$

Jest to liniowe równanie różnicowe o stałych współczynnikach, o ogólnym rozwiązaniu postaci

$$y_j = \sum_{k=0}^{m} c_k \beta_k^j, \tag{6.10}$$

gdzie c_k są stałymi, wyznaczonymi przez warunki początkowe, a β_k są (z założenia różnymi) pierwiastkami równania charakterystycznego

$$(1 - h\lambda b_{-1})\beta^{m+1} = \sum_{k=0}^{m} (a_k + h\lambda b_k)\beta^{m-k}. \tag{6.11}$$

Jeżeli takie rozwiązanie ma aproksymować rozwiązanie równania różniczkowego, to jeden z pierwiastków, powiedzmy β_0, powinien być w przybliżeniu równy $e^{\lambda h}$. Dla $h \ll 1$ możemy rozwinąć ten pierwiastek w szereg Taylora

$$\alpha_0 = 1 + \sum_{n=1}^{\infty} \beta_n h^n. \tag{6.12}$$

Wstawiając to rozwinięcie do równania charakterystycznego (6.11) można pokazać, że $\beta_1 = \lambda$. Jeżeli metoda całkowania numerycznego jest kosystentna, to

$$\alpha_0 = e^{\lambda h} + O(h^2). \tag{6.13}$$

Rozwiązanie równania różnicowego aproksymuje odpowiednie rozwiązanie równania różniczkowego o ile $c_0 \approx y_0$ i pozostałe wyrazy są dostatecznie małe.

6.2 Metody Rungego-Kutty

Metody Rungego-Kutty to metody jednokrokowe - używają informacji tylko z poprzedniego kroku. Dokładność wyższego rzędu jest osiągana przez rozważanie punktów pośrednich. Przykładem takiej metody jest

$$y_{n+1} = y_n + \frac{1}{2} h_n \left[f(t_n, y_n) + f(t_n + h_n, y_n + h_n f_n) \right], \tag{6.14}$$

gdzie $f_n = f(t_n, y_n)$ i $h_n = t_{n+1} - t_n$ jest długościa n-tego kroku. W ogólności metody Rungego-Kutty definuje się następująco.

$$y_{n+1} = y_n + \sum_{i=1}^{m} w_i k_i, \tag{6.15}$$

gdzie w_i są stałymi i

$$k_i = h_n f \left(x_n + \alpha_i h_n, y_n + \sum_{j=1}^{i-1} \beta_{ij} k_j \right). \tag{6.16}$$

Parametry α_i i β_{ij} są stałymi ($\alpha_1 = 0$). Ich wartości są ustalane, tak aby metoda osiągała wybrany rząd dokładności. W tym celu rozwijamy obie strony w szereg Taylora w punkcie (t_n, y_n) i dopasowujemy wyrazy do założonego rzędu. Na przykład możemy napisać

$$\begin{aligned} y_{n+1} - y_n &= h_n y'_n + \frac{1}{2}h_n^2 y''_n + \frac{1}{6}h_n^3 y'''_n + \dots \\ &= h_n f + \frac{1}{2}h_n^2 (f_t + f f_y) + \dots, \end{aligned} \tag{6.17}$$

gdzie $f = f(t_n, y_n)$ oraz f_t i f_y są pochodnymi cząstkowymi obliczonymi w tym samym punkcie. Aby otrzymać formułę drugiego rzędu musimy dopasować te trzy wyrazy do wyrazów formuły (6.15) z $m = 2$, danych przez

$$\begin{aligned} y_{n+1} - y_n &= h_n \left[w_1 f(t_n, y_n) + w_2 f(t_n + \alpha_2 h_n, y_n + \beta_{21} h_n f) \right] \\ &= h_n (w_1 f + w_2 f) + h_n^2 w_2 (\alpha_2 f_t + \beta_{21} f f_y) + \dots . \end{aligned} \tag{6.18}$$

Dokładność drugiego rzędu oznacza, że te dwa rozwinięcia zgadzają się do wyrazów zawierających h_n^2. Porównując współczynniki przy f, f_t i $f f_y$ w (6.17) i (6.18) otrzymujemy równania określajace współczynniki w_1, w_2, α_2 i β_{21}

$$\begin{aligned} w_1 + w_2 &= 1, \\ w_2 \alpha_2 &= \frac{1}{2} \\ w_2 \beta_{21} &= \frac{1}{2}. \end{aligned} \tag{6.19}$$

Te trzy nieliniowe równania z czterema niewiadomymi, rozwiązuje się dla w_1, w_2 i β_{21}, zakładając, że α_2 jest znane. Najczęściej używane metody drugiego rzędu odpowiadają $\alpha_2 = \frac{1}{2}$

$$y_{n+1} = y_n + h_n f\left(t_n + \frac{1}{2}h_n, y_n + \frac{1}{2}h_n f_n\right), \tag{6.20}$$

$\alpha_2 = \frac{2}{3}$

$$y_{n+1} = y_n + \frac{1}{4}h_n \left[f(t_n, y_n) + 3f\left(t_n + \frac{2}{3}h_n, y_n + \frac{2}{3}h_n f_n\right)\right] \tag{6.21}$$

i $\alpha_2 = 1$ metoda dana formułą (6.14).

Uwzględniając wyrazy wyższych rzędów otrzymamy formuły wyższych rzędów, w tym szeroko używaną metodę klasyczną metodę czwartego rzędu

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ k_1 &= h_n f(t_n, y_n), \\ k_2 &= h_n f(t_n + \frac{1}{2}h_n, y_n + \frac{1}{2}k_1), \\ k_3 &= h_n f(t_n + \frac{1}{2}h_n, y_n + \frac{1}{2}k_2), \\ k_4 &= h_n f(t_n + h_n, y_n + k_3). \end{aligned} \tag{6.22}$$

Metody rzędu $r \leqslant 4$ wymagają obliczenia r wartości funkcji f na krok. Między innymi formuła (6.22) została zimplementowana w MATLAB-owskiej procedurze *ode*45, rozwiązujacej zagadnienia początkowe dla układów równań różniczkowych zwyczajnych.