Zespół Szkół Mechanicznych
im. Komisji Edukacji Narodowej
Poznań

Michał Martyński

Możliwości pakietu Scilab

SPIS TREŚCI

1. Wstęp

Pakiet Scilab [1] to wszechstronne narzędzie. W niniejszej pracy przedstawiono podstawy obsługi pakietu oraz jego zastosowanie do estymowania parametrów za pomocą nieliniowej metody najmniejszych kwadratów [2] na przykładzie prostych rozkładów prawdopodobieństwa. Zaprezentowane skrypty po niewielkich modyfikacjach można łatwo wykorzystać do symulacji i analizy zjawisk na zaawansowanym poziomie np. do wyznaczania fizykochemicznych parametrów w spektroskopii pojedynczych cząsteczek [3].

Estymowanie parametrów jest jednym z podstawowych procedur stosowanych w wielu dziedzinach nauki i techniki. Już studenci pierwszych lat studiów, takich kierunków jak fizyka, chemia, czy ekonomia, do opracowania wyników ćwiczeń laboratoryjnych powszechnie stosują liniową metodę najmniejszych kwadratów do wyznaczania parametrów [4, 5]. Już w tak prostych przypadkach potrzebne jest narzędzie służące do analizy danych. Do opracowania niewielkiej ilości danych można zastosować arkusz kalkulacyjny, jeden z wielu prostych, niedużych programów dostępnych za darmo w Internecie, czy nawet bardziej rozbudowany kalkulator, lub przeprowadzić obliczenia na kartce papieru. Jeśli jednak modelu do którego dopasowujemy parametry nie opisuje funkcja liniowa, a ponadto zachodzi potrzeba analizy setek, czy tysięcy krzywych warto poszukać wielofunkcyjnego narzędzia pozwalającego zautomatyzować obliczenia. W tej pracy do tego celu proponowany jest pakiet służący do obliczeń numerycznych – Scilab. Jest to potężne narzędzie, dające możliwości bardzo różnorakich zastosowań. Jednym z nich jest estymacja parametrów za pomocą nieliniowej metody najmniejszych kwadratów. Pakiet ten pozwala też na generowanie danych metodą Monte Carlo, a także umożliwia atrakcyjną wizualizację uzyskanych wyników. Wydaje się wiec być godnym zainteresowania, choć nie jedynym i nie pozbawionym wad, narzędziem do rozwiązywania problemów rozważanych w tej pracy.

2 Pakiet Scilab

W rozdziale tym znajdą się podstawowe informacje o pakiecie Scilab. Omówione zostaną wybrane możliwości Scilaba wraz z przykładami. Informacje zawarte tutaj mogą okazać się pomocne dla osób rozpoczynających pracę z pakietem.

2.1 Co to jest Scilab?

Scilab jest niekomercyjnym, uniwersalnym pakietem obliczeniowym umożliwiającym wizualizację otrzymanych wyników. Jest to darmowy odpowiednik pakietu Matlab [6]. Powstanie Matlaba sięga lat siedemdziesiątych. Jego nazwa pochodzi od słów Matrix Laboratory, gdyż pierwotnym przeznaczeniem programu były numeryczne obliczenia macierzowe. Scilab nie jest jedynym darmowym klonem Matlaba. Popularnym pakietem tego typu jest również Octave [7].

Scilab został stworzony w 1990 roku przez francuski instytut INRIA (Institut National de Recherche en Informatique et Automatique) oraz ENPC (Ecole Nationale des Ponts et Chauss´ees). Należy on do grupy programów open source, od 1994 roku jest bezpłatnie dostępny w Internecie [1]. Od 2003 roku rozwijany jest przez utworzone specjalnie do tego celu Scilab Consortium. Pakiet dostępny jest na wiele systemów operacyjnych. Najnowsza wersja Scilaba oferowana jest pod Linux, Windows i MacOS.

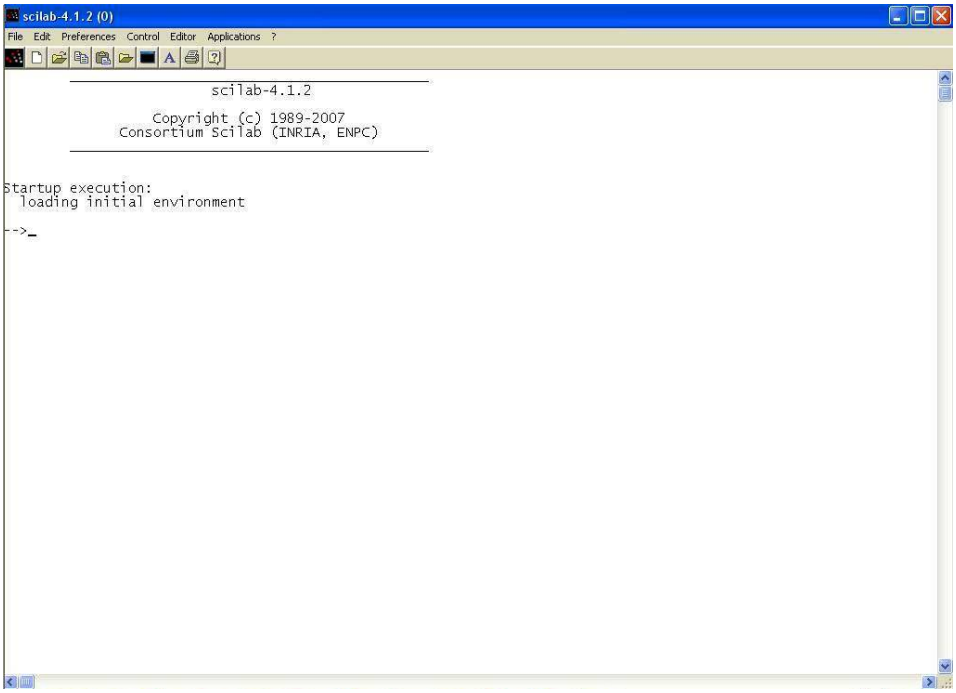
Scilab umożliwia wykonywanie różnorodnych obliczeń matematycznych. Pozwala między innymi na: rozwiązywanie układów równań, równań różniczkowych, obliczenia na wielomianach, wektorach i macierzach, generowanie liczb pseudolosowych, tworzenie wykresów dwu- i trójwymiarowych, operacje na plikach. Scilab może również pełnić funkcje powłoki systemowej. Skrypty przedstawione w niniejszej pracy pokazują możliwości wykorzystaniu pakietu do symulacji, estymacji parametrów, a także wizualizacji wyników.

Scilab posiada własny skryptowy język programowania wysokiego poziomu. Jest on zbliżony do innych języków (Fortran, Basic, C++), wiec osoba posiadająca podstawy programowania nie powinna mieć problemów w szybkim zorientowaniu się w jego składni. Pakiet umożliwia stworzenie interfejsu użytkownika, co pozwala na korzystanie ze

skryptów bez znajomości i konieczności ingerencji w kod programu. Skrypty prezentowane w niniejszej pracy wyposażone są w interfejs użytkownika.

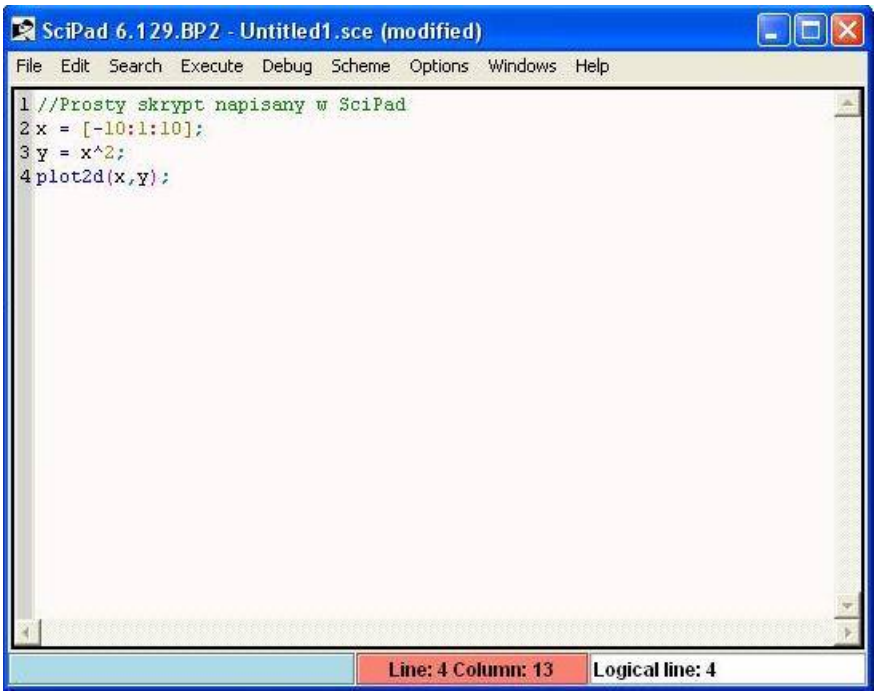
Problemem w pracy z Scilabem może okazać się niewielka ilość materiałów dydaktycznych w szczególności w języku polskim. Na rynku dostępne są dwie pozycje książkowe [8, 9], lecz tylko jedna z nich skupia się wyłącznie na Scilabie. Pomocne może okazać się wprowadzenie do pakietu autorstwa Bruno Pinçon'a w tłumaczeniu Piotra Fulmańskiego i Katarzyny Szulc dostępne bezpłatnie w sieci [10]. Można znaleźć również skromniejsze materiały na stronach uczelni wyższych np. instrukcje dla studentów do zajęć prowadzonych z wykorzystaniem pakietu. Rozwiązania konkretnych problemów można szukać na polskojęzycznym forum Matlaba, zawierającym podforum dotyczące klonów programu [11]. Zbiór darmowych publikacji znajduje się na stronach [12 – 14]. Znaleźć tam można między innymi obszerne opracowanie rosyjskojęzyczne [15]. Warto skorzystać z pomocy do pakietu dostępnej w językach angielskim i francuskim. Pomoc zawiera, poza opisem funkcji, praktyczne przykłady. Można sięgnąć po materiały przeznaczone dla Matlaba protoplasty Scilaba. Języki tych pakietów nie są jednak tożsame, wiec zastosowanie tych materiałów jest ograniczone. Liczba publikacji opisujących ten komercyjny pakiet jest nieporównywalnie większa. Również porównanie wydajności [16] pokazuje przewagę Matlaba nad jego niekomercyjnymi klonami (Scialb, Octave). Co przemawia wiec na korzyść Scilaba i dlaczego warto się nim zainteresować mimo bezsprzecznej przewagi Matlaba w niektórych dziedzinach? Głównym i dość mocnym argumentem są pieniądze. Edukacyjna wersja Matlaba to wydatek ok. 3 tysięcy złotych [17], a dodatkowe pakiety również są płatne.

Scilab zawiera kilka komponentów. Po uruchomieniu programu pojawia się okno konsoli (rys. 1). Można wpisywać polecenia bezpośrednio na konsoli, tutaj będą też widoczne efekty działania uruchamianych skryptów.



Rys. 1 Okno konsoli

Rzadko stosuje się konsolę do wpisywania poleceń ponieważ efekty znikają wraz z wyłączeniem programu. Dużo większe możliwości daje stworzenie skryptu, z którego po zapisaniu można wielokrotnie korzystać. Do pisania skryptów dostępne jest w pakiecie użyteczne narzędzie edytor SciPad (rys. 2).



Rys. 2 Okno edytora SciPad

SciPad rozpoznaje składnie i koloruje słowa kluczowe. Bardzo pomocna jest też numeracja lini. Z poziomu SciPada można bezpośrednio uruchomić skrypt z menu przez *Execute → Load into Scilab*, lub stosując skrót klawiaturowy *Ctrl+l* Skrypty nie są kompilowane przed wykonaniem, tylko wykonywane linia po linii. Skutkuje to spadkiem wydajności w porównaniu z kompilowanym kodem (np. Fortran, C++) nie ma jednak potrzeby deklarowania typów zmiennych, czy rozmiarów tablic i mogą się one zmieniać dynamicznie podczas wykonywania skryptu. Program można więc uruchomić nawet gdy zawiera błędy w kodzie, co nie jest możliwe w przypadku kodu wymagającego kompilacji gdzie błędy zostają wykryte podczas kompilacji, a wykonanie możliwe jest dopiero po ich usunięciu. W przypadku Scilaba skrypt wykona się do miejsca w którym występuje błąd. Informacja o błędzie i numer linii kodu, w którym został zlokalizowany zostaną wypisane na konsoli.

SciPad posiada możliwość włączenia polskiej wersji językowej (*Options → Lokale → Polski (pl))*. Tłumaczenie nie jest jednak doskonałe. Często, szczególnie głębsze opcje menu nie są dostępne w języku polskim. Pozostałe składniki pakietu nie oferują polskiej wersji językowej. W całości Scilab jest dostępny w językach angielskim i francuskim.

Jedną z możliwości Scilaba jest wizualizacja otrzymanych wyników. W tym calu wywoływane jest okno graficzne. Posiada ono menu pozwalające na podstawową edycje rysunku.

2.2 Podstawy pracy z pakietem Scilab

Największą trudnością na początku przygody z Scilabem jest poznanie składni języka, którym pakiet dysponuje. Osoby mające podstawową wiedzę na temat programowania szybko powinny sobie z tym poradzić. Ten rozdział ma na celu umożliwienie rozpoczęcia pracy z pakietem poprzez zaprezentowanie podstawowego zestawu poleceń oraz przedstawienie przykładów praktycznego zastosowania omawianych funkcji.

2.2.1 Scilab jako kalkulator

Najprostszym zastosowaniem Scilaba są działania matematyczne, które jest w stanie wykonać zwykły kalkulator. Operatory działań matematycznych znajdują się w tabeli 2.1.

Tabela 2.1 Podstawowe działania matematyczne

+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
^	potęgowanie
sqrt	pierwiastkowanie

Aby niniejszy tekst pozwalał na łatwe rozpoczęcie pracy z pakietem zostaną przedstawione praktyczne przykłady opisywanych operacji.

Przykład 2.1

```
-->2+3
 ans  =
    5.
```

Jak widać przeprowadzanie prostych działań matematycznych jest bardzo łatwe i intuicyjne. Najczęściej wartości przypisujemy zmiennym (przykład 2.2). Jeśli nie przypiszemy zmiennej program automatycznie przypisuje wartość do zmiennej *ans*, co pokazuje przykład 2.1

Przykład 2.2

```
-->a=2
 a  =
    2.
-->b      =       3
 b  =
    3.
```

```
-->c = a * b
 c  =
   6.
```

W powyższym przykładzie zaobserwować można, ze użycie spacji nie zmienia wyniku wykonywanej operacji. Takie znaki jak spacja możemy stosować dowolnie poprawiając czytelność kodu.

Przeglądając kody skryptów Scilaba (rozdział 3.3) zaobserwować można, że bardzo wiele linii zakończonych jest średnikiem. Użycie średnika powoduje to, że wynik działania danej linii kodu nie zostanie wyświetlony na konsoli.

Przykład 2.3

```
-->a = 2;
-->b      =       3;
-->c = a * b
 c  =
   6.
```

Pomocna jest możliwość wstawiania komentarzy. Komentarz to fragment kodu, którego program nie analizuje. Istnieje możliwość zakomentowania całego bloku programu. Fragment kodu należy zaznaczyć i użyć skrótu klawiaturowego *ctrl* + *m*, zlikwidowanie komentarza następuje poprzez użycie skrótu *ctrl* + *M*. Wyżej wymienione czynności można oczywiście również wykonać korzystając z menu. Zastosowanie komentarza pokazuje przykład 2.4

Przykład 2.4

```
-->b = 2/3 //to jest komentarz
 b  =
   0.6666667
```

Tytuł niniejszego podrozdziału brzmi „Scilab jako kalkulator”. Rozbudowane kalkulatory pozwalają wykonywać bardziej zaawansowane działania np. obliczać wartości funkcji trygonometrycznych, czy logarytmów. Oczywiście dla pakietu Scilab takie zadania nie stanowią najmniejszego problemu. W tabeli 2.2 znajdują się wybrane, często stosowane

funkcje, możliwe do zaimplementowania w Scilabie. Natomiast w tabeli 2.3 zebrane zostały wybrane predefiniowane zmienne.

Tabela 2.2 Wybrane funkcje dostępne w pakiecie Scilab

sin	sinus
cos	kosinus
tan	tangens
cotg	kotangens
exp	eksponent
log	logarytm naturalny
log10	logarytm dziesiętny
log*b*	logarytm o podstawie *b*
abs	wartość bezwzględna
modulo	reszta z dzielenia
int	ignoruje część ułamkową liczby
floor	zaokrąglenie w dół do liczby całkowitej
ceil	zaokrąglenie w górę do liczby całkowitej
format	dokładność wypisywania liczb
clear	czyszczenie pamięci
rand	generator liczb pseudolosowych
mean	wartość średnia
stdev	odchylenie standardowe
min	wartość minimalna
max	wartość maksymalna

Tabela 2.3 Wybrane zmienne predefiniowane

%pi	stała
%e	podstawa logarytmu naturalnego
%i	jednostka urojona
%inf	nieskończoność
%eps	minimalna możliwa do zapisania w danym systemie liczba zmiennoprzecinkowa

%t	prawda
%f	fałsz

Poniżej znajduje się kilka przykładów użycia funkcji z tabeli 2.2.

Wszystkie funkcje trygonometryczne wymagają jako parametru wartości konta podanego w radianach.

Przykład 2.5

```
-->a = sin(%pi/2)
 a  =
    1.
```

Scilab pozwala obliczyć wartość logarytmu o dowolnej podstawie

Przykład 2.6

```
-->log2(1024)
 ans  =
    10.
```

Funkcja *modulo*(*a*,*b*) zwracająca resztę z dzielenia potrzebuje dwóch argumentów: *a* – dzielna, *b* – dzielnik.

Przykład 2.7

```
-->modulo(2,3)
 ans  =
    2.
```

Przykład użycia funkcji statystycznych znajduje się w skrypcie o kodzie 3.3 (rozdział 3.3).

2.2.2 Macierze

Podstawowym typem danych jakim posługuje się Scilab jest macierz. Nawet przypisując wartość skalarna do zmiennej (Przykład 2.2) tak naprawdę przez Scilaba

widziana jest ona jako macierz o wymiarze jednego wiersza i jednej kolumny. Przekonać się o tym można stosując funkcje *size* zwracającą wielkość zmiennej.

Przykład 2.8

```
-->a = 5
 a  =
   5.
-->size(a)
 ans  =
   1.   1.
```

Dzięki temu operacje na macierzach są bardzo proste do zaimplementowania w Scilabie. W większości popularnych języków programowania wypełnienie macierzy trzeba przeprowadzić przy pomocy pętli. W omawianym pakiecie jest to dużo prostsze. Jednym ze sposobów definiowania macierzy jest umieszczenie jej elementów w nawiasach kwadratowych. Elementy tego samego wiersza oddzielają spacja, lub przecinek, natomiast wiersze kończą się średnikiem.

Przykład 2.9

```
-->A = [2, 4, 8; 3, 9, 27; 4, 16, 64]
 A  =
   2.   4.    8.
   3.   9.    27.
   4.   16.   64.
```

Dla czytelności można umieścić każdy z wierszy w osobnej linii.

Przykład 2.10

```
-->A = [2 4 8;
-->     3 9 27;
-->     4 16 64]
 A  =
   2.   4.    8.
   3.   9.    27.
```

```
   4.    16.    64.
```

Bardzo proste jest definiowanie wektorów, będących szczególnymi przypadkami macierzy. *a*:*h*:*b* da wektor o wartości początkowej *a*, końcowej *b* i utworzony on będzie z krokiem *h*. Jeśli krok wynosi 1 można go pominąć.

Przykład 2.11

```
x = 10:2:16
 x  =
   10.    12.    14.    16.
```

Pomocną funkcją przy definiowaniu wektorów jest *linspace*(*a*,*b*,*n*). Tworzy ona wektor posiadający *n* równomiernie rozłożonych elementów pomiędzy *a* i *b*.

Przykład 2.12

```
x = linspace(0,1,11)
 x  =
   0.    0.1    0.2    0.3    0.4    0.5    0.6    0.7    0.8    0.9    1.
```

By zmienić wektor wierszowy na kolumnowy używa się apostrofu.

Przykład 2.13

```
-->x = 8:4:20
 x  =
   8.    12.    16.    20.

-->x'
 ans  =
   8.
   12.
   16.
   20.
```

Operacja ta możliwa jest również dla macierzy. Macierz A w przykładzie 3.14 zdefiniowana została w przykładzie 3.10

Przykład 2.14

```
-->A'
 ans  =
   2.    3.    4.
   4.    9.    16.
   8.    27.   64.
```

Za pomocą funkcji *rand* można wygenerować macierz wypełnioną liczbami losowymi

Przykład 2.15

```
B = rand(2,3)
 B  =
   0.2113249    0.0002211    0.6653811
   0.7560439    0.3303271    0.6283918
```

W tabeli 2.4 zebrano funkcje służące do generowania typowych macierzy.

Tabela 2.4 Funkcje służące do generowania macierzy

eye	macierz jednostkowa
diag	macierz diagonalna
ones	macierz jedynkowa
zeros	macierz zerowa
triu	macierz trójkątna górna
tril	macierz trójkątna dolna

Prosto można się odwołać do konkretnego elementu macierzy oraz wydobyć podmacierz z macierzy głównej

Przykład 2.16

```
-->A
```

```
 A  =
    2.    4.     8.
    3.    9.     27.
    4.    16.    64.

-->A(1,3) //element macierzy A
 ans  =
    8.

-->A(:,3) //trzecia kolumna macierzy A
 ans  =
    8.
    27.
    64.

-->A(2,:) //drugi wiersz macierzy A
 ans  =
    3.    9.    27.

-->A([1,3],[2,3]) //podmacierz macierzy A
 ans  =
    4.     8.
    16.    64.
```

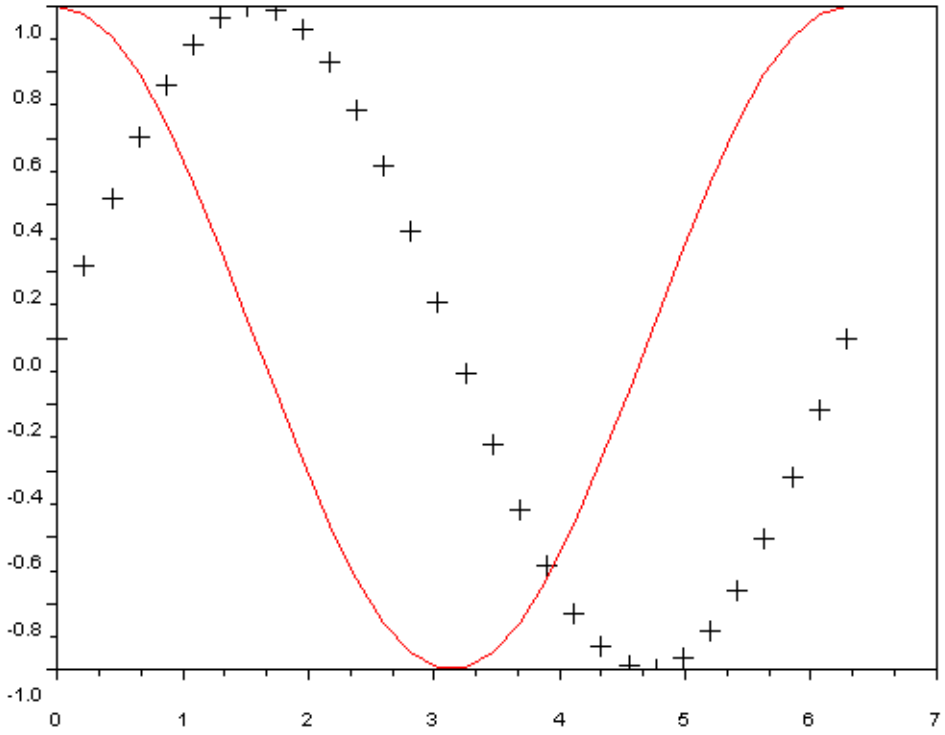
2.2.3 Wykresy

Wielką zaletą Scilaba są bogate możliwości wizualizacji wyników. W tym rozdziale omówione zostaną podstawowe procedury pozwalające na tworzenie wykresów dwu- i trójwymiarowych, histogramów oraz wykresów panelowych.

Najprostszym sposobem stworzenia wykresu jest zastosowanie instrukcji *plot*. Jednak dużo większe możliwości daje *plot2d*. Zastosowanie tej instrukcji do stworzenia wykresu funkcji sinus i cosinus przedstawione jest w przykładzie 2.17.

Przykład 2.17

```
-->x = linspace (0,2*%pi,30);
-->plot2d(x,sin(x),-1)
-->plot2d(x,cos(x),5)
```



Rys. 3 Zastosowanie instrukcji *plot2d*

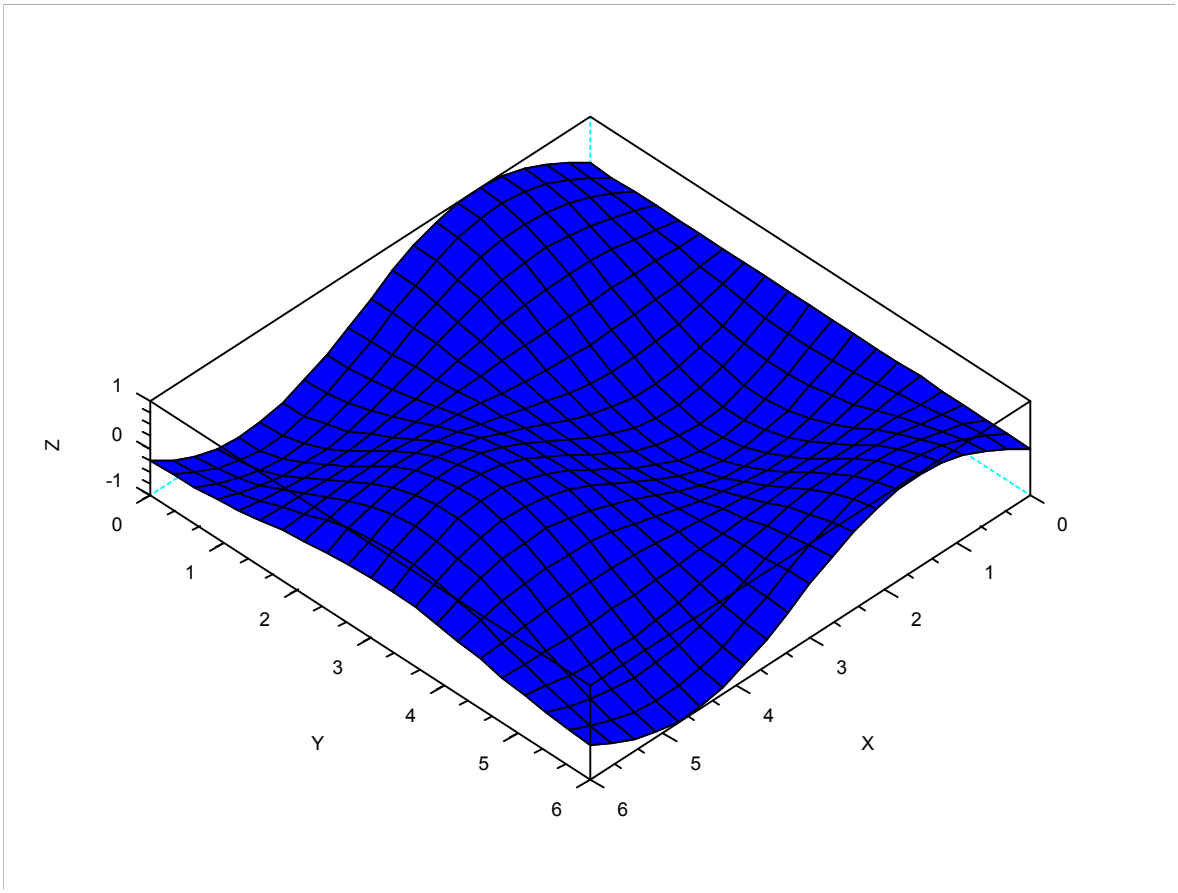
Instrukcja *plot2d* wymaga dwóch argumentów – wartości *x* i *y*. Pozostałe parametry są opcjonalne. Domyślnie powstaje wykres liniowy, linia ma kolor czarny. W przykładzie 2.17 zmieniony został typ wykresu funkcji sinus na punktowy i wybrane wyświetlanie punktów w postaci znaków „+”, a kolor linii przedstawiający drugą zależność zmieniono na czerwony. Możliwych opcji jest dużo więcej. Można ustalać zakres skali na osiach, stosować skalę logarytmiczną, dodawać legendę itp. Wszystkie możliwości instrukcji *plot2d* znaleźć można w pomocy pakietu.

Przykład 2.18 pokazuje sposób tworzenia wykresów trójwymiarowych z zastosowaniem instrukcji *plot3d*.

Przykład 2.18

```
-->t=[0:0.3:2*%pi]';
```

```
-->z=sin(t)*cos(t');
-->plot3d(t,t,z)
```

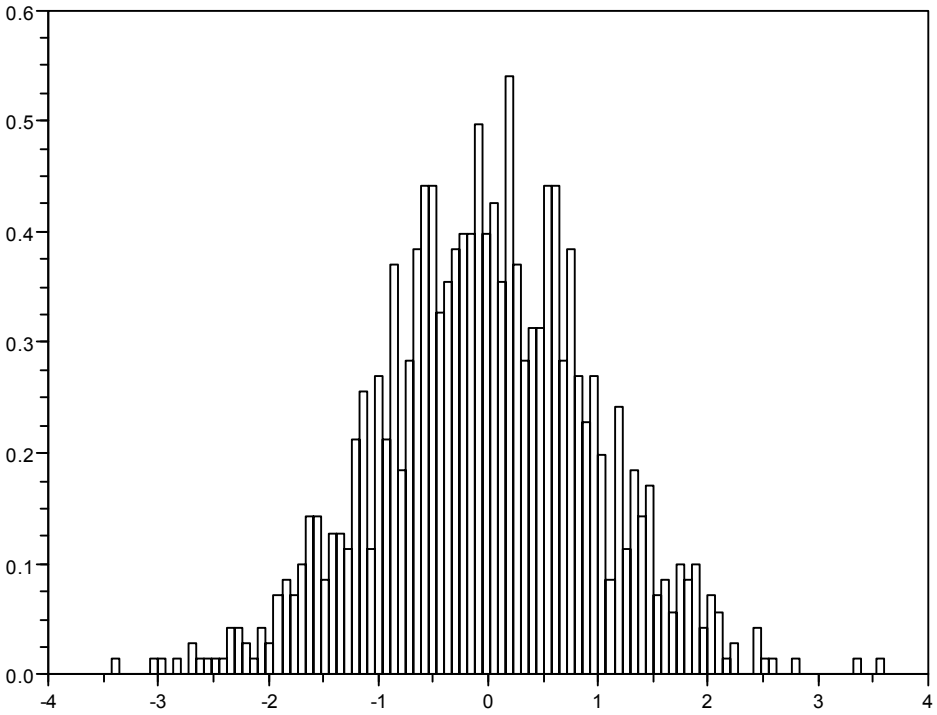


Rys. 4 Zastosowanie instrukcji *plot3d*

Scilab jest dobrym narzędziem do tworzenia histogramów. Służy do tego instrukcja *histplot*. Niestety o ile pozwala ona na łatwe sporządzenie wykresu, to nie daje dostępu do wartości liczbowych stworzonego histogramu. Należy to uznać za sporą wadę. Analogiczna instrukcja Matlaba nie posiada tej niedogodności. W najprostszej wersji *histplot* wymaga dwóch argumentów – wielkości które mają zostać histogramowane i liczbę klas w których mają być rozmieszczone. W kolejnym przykładzie tworzony jest histogram z liczb losowych wygenerowanych z rozkładu normalnego.

Przykład 2.19

```
-->y = rand(1,1000,'normal');
-->histplot(100,y)
```

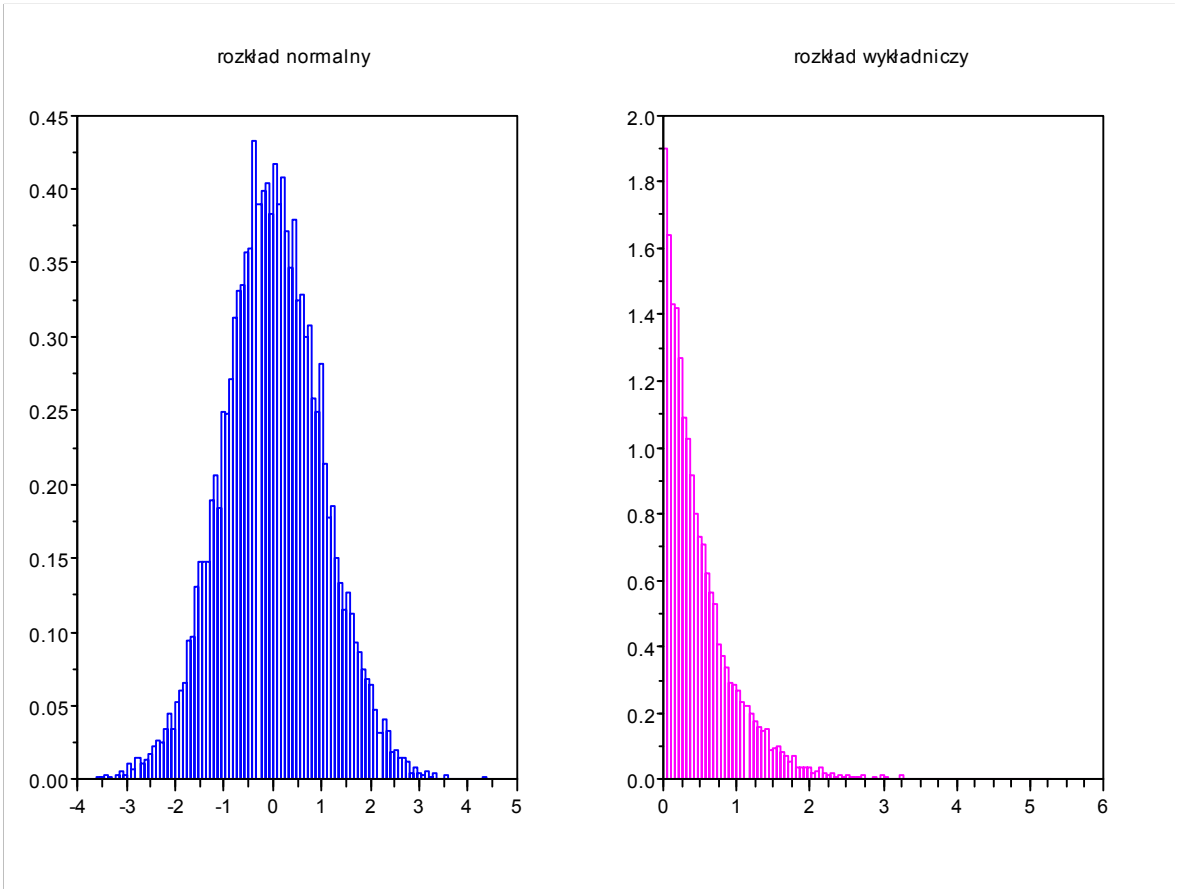


Rys. 5 Przykład histogramu

Ciekawym sposobem wizualizacji jest tworzenie wykresów panelowych. Odpowiada za to instrukcja *subplot*. Definiuje się w niej ilość wierszy i kolumn w wykresie panelowym oraz numer bieżącego panelu w którym ma być umieszczony wykres.

Przykład 2.20

```
subplot(1,2,1)
y = rand(1,10000,'normal');
histplot(100,y,2)
xtitle('rozkład normalny')
subplot(1,2,2)
x = rand(1,10000);
y = -log(1-x)/2;
histplot(100,y,6)
xtitle('rozkład wykładniczy')
```



Rys. 6 Przykład wykresu panelowego

2.2.4 Programowanie

Scilab dysponuje własnym językiem programowania. Jest on stosunkowo prosty do opanowania. Wpływa na to między innymi brak konieczności deklarowania typów i wielkości zmiennych. O ile programowanie w Scilabie jest łatwiejsze w porównaniu z programowaniem w językach wymagających kompilacji kodu (np. Fortran, C++), to wydajność jego skryptów jest mniejsza.

Trudno wyobrazić sobie napisanie funkcjonalnego skryptu bez pętli, czy instrukcji warunkowych. Język Scilaba dysponuje tymi instrumentami, a ich składnia jest podobna do tej występującej w innych językach programowania. W tym rozdziale zaprezentowane zostaną pętle, instrukcje warunkowe i możliwość tworzenia własnych funkcji przez użytkownika.

2.2.4.1 Instrukcje warunkowe

Instrukcje warunkowe służą do podejmowanie przez program decyzji. Najpopularniejszą tego typu instrukcją jest *if*. Składnia instrukcji *if* wygląda następująco:

if warunek 1 *then*
    instrukcje wykonywane gdy spełniony jest warunek 1
elseif warunek 2 *then*
    instrukcje wykonywane gdy spełniony jest warunek 2
elseif warunek n *then*
    instrukcje wykonywane gdy spełniony jest warunek n
else
    instrukcje wykonywane gdy nie jest spełniony żaden z warunków od 1 do n
end

Do formułowania warunków potrzebne są operatory porównania. Zostały one zabrane w tabeli 2.5. Warto zwrócić uwagę na to, że operator „=" jest operatorem przypisania, jeśli program ma sprawdzić równość dwóch wartości należy zastosować operator „=="

Tabela 2.5 Operatory porównania

==	równe
~= lub <>	różne
>	większe
<	mniejsze
>=	większe lub równe
<=	mniejsze lub równe

Można budować też warunki złożone stosując operatory logiczne: „&" i; „|" – lub; „~" – nie. Przykłady praktycznego wykorzystania instrukcji warunkowej można znaleźć w skryptach przedstawionych w rozdziale 3. Inną instrukcją warunkową jest *select case*, nie będzie ona tutaj omawiana.

2.2.4.2 Pętle

Jeśli program ma powtarzać pewien ciąg instrukcji należy zastosować pętle. Można skorzystać z pętli *for*, lub *while*. Składnia pętli *for* jest następująca:

for i:*h*:*n*

ciąg instrukcji

end

Zmienna *i* jest w tym przypadku licznikiem pętli i zmienia się z krokiem *h* do *n*. Na skutek działania pętli instrukcje będące pomiędzy *for* i *end* będą powtarzane.

Podobnie działa funkcja *while*.

While warunek

ciąg instrukcji

end

Powtarza ona instrukcje będące we wnętrzu pętli dopóki spełniony jest podany warunek.

2.2.4.3 Funkcje

Scilab umożliwia definiowanie własnych funkcji przez użytkownika. Funkcje otwiera słowo kluczowe *function*, a zamyka *endfunction*, pomiędzy nimi znajduje się ciało funkcji, czyli instrukcje, które są przez nią wykonywane.

Function [*b*1, *b*2, …, *b*n] = *nazwa_funkcji* (*a*1, *a*2, …, *a*n)

ciało funkcji

endfunction

Parametry *a* są pobierane przez funkcje, natomiast *b* są parametrami przez funkcje zwracanymi. Zdefiniowanie funkcji pozwala wywoływać ją wielokrotnie w programie. Funkcja może zostać zapisana w pliku i stosowana w różnych skryptach. Zwyczajowo skrypty zapisuje się z rozszerzeniem *sce*, a funkcje *sci*, lecz pomiędzy tymi rozszerzeniami tak naprawdę nie ma żadnej różnicy. Aby korzystać z funkcji znajdującej się w innym pliku trzeba ją wczytać przez:

getf(″nazwa_pliku.sci″)

Poniżej znajduje się przykładowa funkcja obliczająca pierwiastek trzeciego stopnia.

Przykład 2.21

```
function p = pierwiastek3(a)
    p = a^(1/3);
endfunction
```

wywołanie funkcji:

```
c = pierwiastek3(64)
```

wynik:

```
→ c  =
   4.
```

3. Estymacja parametrów za pomocą pakietu Scilab

W rozdziale tym zostaną zaprezentowane skrypty Scilaba służące do generowania danych i estymacji parametrów. Omówione zostaną również krótko rozkłady prawdopodobieństwa, z których losowane są dane oraz metoda najmniejszych kwadratów.

3.1 Rozkłady prawdopodobieństwa

Prezentowane w rozdziale trzecim skrypty mają na celu obrazować możliwości pakietu Scilab. Dlatego dane generowane są z popularnych rozkładów prawdopodobieństwa. Wybrano dwa rozkłady prawdopodobieństwa zmiennych losowych ciągłych: normalny i wykładniczy.

3.1.1 Rozkład normalny

Rozkład normalny nazywany równie często rozkładem Gaussa jest jednym z ważniejszych rozkładów prawdopodobieństwa. Jeśli jakaś wielkość jest sumą lub średnią bardzo wielu drobnych losowych czynników, to niezależnie od rozkładu każdego z tych czynników, jej rozkład będzie zbliżony do normalnego.

Zmienna losowa gaussowska to zmienna losowa ciągła określona na przedziale $(-\infty, \infty)$ o gęstości:

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{\left[-\frac{(x-m)^2}{2\sigma^2}\right]} \tag{1}$$

Parametrami rozkładu są:

- m – wartość oczekiwana
- $\sigma > 0$ – wariancja

Wartość oczekiwana m nie zmienia kształtu rozkładu, a jedynie położenie na osi x. Kształt krzywej zmienia się wraz ze zmianą wariancji σ. Całkowite pole powierzchni pod krzywą jest równe 1.

Generowanie liczb z rozkładu normalnego umożliwia już najprostszy generator liczb pseudolosowych Scilaba (przykład 2.19)

3.1.2 Rozkład wykładniczy

Rozkład wykładniczy opisuje sytuację, w której obiekt może przyjmować stany A i B oraz może ze stałym prawdopodobieństwem przejść ze stanu A w stan B w jednostce czasu. Prawdopodobieństwo wyznaczane przez ten rozkład to prawdopodobieństwo przejścia ze stanu A w B w czasie Δt. Rozkładem tym można opisać wiele zjawisk fotofizycznych np. zmiany konformacyjne cząsteczki, lub zmiany stanów energetycznych cząsteczki (singlet – tryplet)

Zmienna losowa wykładnicza to zmienna losowa określona na przedziale $(0, \infty)$, której gęstość wyraża się wzorem

$$y = \lambda e^{-\lambda x} \tag{2}$$

gdzie $\lambda > 0$ to parametr rozkładu wykładniczego.
Funkcja Scilaba *rand* nie ma opcji generowania liczb z rozkładu wykładniczego. Można by zastosować bardziej zaawansowany generator liczb pseudolosowych dostępnych w Scilabie – *grand*, który posiada opcje generowania liczb z rozkładu wykładniczego. W skryptach prezentowanych w tej pracy wykorzystano jednak generator *rand*.

Aby przy pomocy wygenerowanych liczb z rozkładu jednostajnego otrzymać rozkład wykładniczy należy skorzystać z twierdzenia o treści: Jeśli zmienna losowa U ma rozkład jednostajny na (0,1), to zmienna $X = F^{-1}(U)$ ma rozkład $F(x)$ [2, 18]
Dystrybuanta wyrażona jest wzorem:

$$F(x) = 1 - e^{-\lambda x} \tag{3}$$

Funkcja odwrotna $x = F^{-1}(y)$ ma postać

$$x = -\ln(1 - y)/\lambda \tag{4}$$

Więc algorytm generowania liczb losowych o rozkładzie wykładniczym wygląda następujaco:

- losowanie liczby u o rozkładzie jednostajnym z przedziału (0,1)
- obliczenie $x = -(\ln u)/\lambda$.

3.2 Metoda najmniejszych kwadratów

Przeprowadzając eksperyment, lub go symulując (jak w niniejszej pracy) na podstawie otrzymanych pomiarów, w postaci par punktów (x_i, y_i) zachodzi potrzeba wyznaczenia parametrów funkcji jaką są one opisane.

$$y = f(x, p) \tag{5}$$

gdzie $p = \{p_1, p_2, \ldots, p_n\}$ są to stałe parametry funkcji, które mają zostać wyznaczone na podstawie punktów pomiarowych. Należy znaleźć takie parametry p, aby punkty wyznaczone po podstawieniu uzyskanych z estymacji p do zależności 5 jak najmniej odbiegały od punktów pomiarowych.

Najpopularniejszą metodą estymacji parametrów jest metoda najmniejszych kwadratów. Bazuje ona na założeniu, ze błędy pomiarowe podlegają rozkładowi normalnemu. Metoda ta dobiera parametry poprzez minimalizację sumy kwadratów odchyleń:

$$S = \sum_{i=1}^{n} (y_i - f(x_i, p) \tag{6}$$

Warunkiem minimalizacji wartości S jest to, by pierwsze pochodne po parametrach p były równe zero.

$$\frac{\partial S}{\partial p_1} = 0, \frac{\partial S}{\partial p_2} = 0, \ldots, \frac{\partial S}{\partial p_n} = 0 \tag{7}$$

W nieliniowej metodzie najmniejszych kwadratów zachodzi potrzeba rozwinięcia funkcji w szereg Taylora. Uwzględnia się jedynie człony liniowe tego rozwinięcia.

W Scilabie wygodna metodą wyznaczania parametrów nieliniowych funkcji jest użycie instrukcji *datafit*. Jest to makrokomenda wywołująca bardziej ogólne polecenie *optim*. Minimalna składnia tej instrukcji jest następująca:

p = *datafit* (*G*, *Z*, *p*0)

Funkcja ta zwraca wyznaczone parametry *p*. *G* to nazwa funkcji definiującej kryterium optymalizacji, *Z* przechowuje punkty pomiarowe do których dopasowywana jest funkcja, a *p*0 są to punkty początkowe dla pierwszej iteracji.

Możliwe jest dodanie wielu opcji do procedury *datafit*. Między innymi dodatkowego kryterium zatrzymania procedury optymalizacyjnej a. maksymalnej liczby iteracji, czy wprowadzenia ograniczeń na dopasowywane parametry.

3.3 Skrypty Scilaba generujące dane i estymujące parametry

W tym rozdziale przedstawione zostaną kody skryptów omawianych w tej pracy. Skrypt o kodzie 3.1 zawiera wyłącznie menu użytkownika pozwalające wybrać rozkład z którego mają być generowane dane. Uruchamia on właściwe skrypty służące do symulacji i estymacji: gauss.sce, lub exp.sce. Skrypty te można uruchamiać też bezpośrednio.

Kod 3.1 dopasowywacz.sce

```
1   //dopasowywacz 1.6.1
2   roz =list('rozkładn',1,['normalny','wykładniczy']);
3   rep=x_choices('Wybierz rozkład z którego chcesz generować dane',list(roz));
4   host('cd>'+TMPDIR+'\path');
5   if rep==1 then
6     exec('gauss.sce');
7   else
8     exec('exp.sce');
9   end
10  buttondialog(„Skonczylem!","ok");
```

Kod 3.2 przedstawia skrypt generujący dane z rozkładu normalnego, umożliwiający estymacje parametrów oraz wizualizację wyników. Po zakomentowaniu fragmentu opisanego jako menu skrypt będzie korzystał z wpisanych z parametrów znajdujących się na początku skryptu. Menu użytkownika będzie wtedy wyłączone. W wyniku działania skryptu tworzony jest plik z wynikami, notowany jest również czas pracy programu oraz warunki symulacji.

By łatwo interpretować wyniki parametry uzyskane w wyniku estymacji zostały znormalizowane w ten sposób, że wartość wyznaczoną podzielono przez tą użytą do

symulacji. Gdyby parametr został wyznaczony idealnie, jego znormalizowana wartość byłaby równa 1. Im bardziej wynik odbiega od jedności tym gorsze są wyniki estymacji.

Skrypt generujący dane z rozkładu wykładniczego ma analogiczną budowę, dlatego nie jest prezentowany w tekscie. Można by pokusić się o zorganizowanie obydwu statystyk w ramach jednego skryptu. Jednak konieczność zastosowania wielu instrukcji warunkowych oraz zmiany nazwy niektórych zmiennych niekorzystnie wpłynęłaby na czytelność kodu.

Kod 3.2 gauss.sce

```
1   //MM v.PP 06.2009
2   //dopasowywacz 1.6.1_gauss
3   ////////////////////////
4   n=10 //liczba symulacji
5   losuj=1 //1 – losowanie pkt.startowych; 0 – zadane pkt. Startowe
6
7   //parametry symulacji
8   m=2.5;
9   sigma=3.0;
10  n_i=100; //rozmiar próby
11  bins=100; //liczba klas histogramu
12
13  //**************menu**************//
14  txt=['liczba symulacji';'m';'sigma'];
15   menu_1=x_mdialog('Liczba symulacji i parametry
16  rozkładu',txt,['100';'2.5';'3']);
17   n=evstr(menu_1(1));
18   m=evstr(menu_1(2));
19   sigma=evstr(menu_1(3));
20
21  m2_1=list('parametry startowe',1,['zadane','losowe']);
22  m2_2=list('wielkość próby',2,['10','10^2','10^3','10^4','10^5','10^6']);
23  menu_2=x_choices(' ',list(m2_1,m2_2));
24  losuj=evstr(menu_2(1));
25  n_i=10^evstr(menu_2(2));
26  //**************menu**************//
27
28  //parametry oryginalne
29  m_org=m;
30  sigma_org=sigma;
31
32  //parametry początkowe
33  m_0=m;
34  sigma_0=sigma;
35
36  ////////////////////////
37  start=getdate('s'); //pomiar czasu
38  fd_wyniki=mopen('gauss_fit.txt','w'); //tworzenie pliku z wynikami
```

```
39  fd_xy=mopen('xy_gauss.txt','w'); // plik zawierajacy symulowane punkty
40  fd_t=mopen('czas.txt','w'); //
41
42  //dopasowywana funkcja
43  function y=FF(x,p),y=exp(-((x-
44  p(1))^2)/(2*p(2)^2))/(sqrt(2*%pi)*p(2)),endfunction
45
46  //kryterium optymalizacji
47  function e=G(p,z),
48    y=z(1),x=z(2);
49    e=(y-FF(x,p)),
50  endfunction
51
52  for k=1:1:n
53    symulacja=k;
54
55    y=sigma*rand(1,n_i,'normal')+m; //generowanie liczb z rozkladu
56
57    //histogramowanie
58    xspace = linspace(min(y), max(y), bins + 1);
59            xsp = xspace(2) – xspace(1);
60            [ind, occ] = dsearch(y, xspace);
61            x = linspace(min(y) + xsp / 2, max(y) – xsp / 2, bins);
62            nor = sum(occ) * xsp;
63            y = occ / nor;
64            Xs = x;
65            Ys = y;
66            Norm = nor;
67    ny_temp = size(y);
68    ny = ny_temp(2);
69
70    //zapisywanie symulacji do pliku
71    for sxy=1:1:ny
72      mfprintf(fd_xy,'%8.4f \t %8.4f \n', x(sxy), y(sxy));
73    end
74
75    //losowanie punktow startowych
76    if losuj==1
77      pktO=rand(1,2);
78      m_0=pktO(1)*(m_org*10-m_org/10)+m_org/10;
79      sigma_0=pktO(2)*(sigma_org*10-sigma_org/10)+sigma_org/10;
80    end
81
82      pg=[m_org;sigma_org]; //parametry oryginalne (uzyte do generacji)
83      p0=[m_0;sigma_0]; //punkty startowe
84      Z=[y;x];
85
86      //wywolanie datafit
87      [p,err]=datafit(G,Z,p0);
88      m_fit = p(1);
```

```
89        sigma_fit = p(2);
90
91        //obliczanie chi-kwadrat
92        chisqr = err/(ny-2);
93
           //zapisywanie wyników
            mfprintf(fd_wyniki,'%1.0f \t %8.4f \t %8.4f \t %8.4f \t %8.4f \t %8.4f \t
94      ....%8.4f \t %8.4f \n', k, m_0, sigma_0, m_fit, sigma_fit, m_fit/m_org,
95      ....sigma_fit/sigma_org, chisqr);
96
97        //tworzenie wykresów
98        scf(1);clf()
99        plot2d(x,FF(x,pg),5)  //oryginalna (red)
100       plot2d3(x,y,1)        //eksperyment
101       plot2d(x,FF(x,p),12)  //dopasowanie (blue)
102
          //wypisywanie informacji na konsole
          mprintf('symulacja= %d, \t m_fit =%8.4f, sigma_fit =%8.4f, chi2 =%8.4f \n',
        ..k, m_fit, sigma_fit, chisqr);
103     end
104
105     //pomiar czasu
106     stop=getdate('s');
107     czas=stop-start
108     tmin=czas/60
109     //info
110     mfprintf(fd_t,'czas = %8.2f \ntmin = %8.2f \nw_proby = %8.2f \nm = %8.2f
        \nsigma = %8.2f', czas, tmin, n_i, m_org, sigma_org);
111
112     //zamkniecie plikow
113     mclose(fd_wyniki);
114     mclose(fd_t);
115     mclose(fd_xy);
116
117     „Skonczylem!"
```

Skrypt o kodzie 3.3 pomaga w analizie wyników. Można za jego pomocą tworzyć histogramy uzyskanych wyników oraz wyznaczyć średnią, odchylenie standardowe, wartość najmniejszą i największa z *n* przeprowadzonych symulacji.

Kod 3.3

```
1     //analiza wynikow
2     function [p_max, p_min, p_mean, p_stdev] = min_max(k)
3       p_max = max(k)
4       p_min = min(k)
5       p_mean = mean(k)
6       p_stdev = stdev(k)
7     endfunction
8
```

```
9   fit_res = mopen('fit_results.txt', 'w');
10
11  //wczytywanie danych
12  dane1 = fscanfMat('g_org.txt');
13  dane2 = fscanfMat('g_los.txt');
14
15  m1 = dane1(:,6);  sigma1 = dane1(:,7);
16  m2 = dane2(:,6);  sigma2 = dane2(:,7);
17
18  //klasy histogramu
19  bins = 50;
20  bins = [0.0:0.1:10.0];
21  //skala
22  a=[0,0,10,6];
23
24   scf(1);
25       subplot(2,2,1) //podzial okna graficznego
26       histplot(bins,m1,style=2,rect=a) //normalization=%f,
27       xtitle („m0 = m_org");
28       subplot(2,2,2)
29       xtitle („sigma0 = sigma_org");
30       histplot(bins,sigma1,style=2,rect=a)
31       subplot(2,2,3)
32       histplot(bins,m2,style=2,rect=a)
33       xtitle („m0 – losowe");
34       subplot(2,2,4)
35       histplot(bins,sigma2,style=2,rect=a)
36       xtitle („sigma0 – losowe");
37
38  [m1_max, m1_min, m1_mean, m1_stdev] = min_max(m1);
39  [m2_max, m2_min, m2_mean, m2_stdev] = min_max(m2);
40
41  [sigma1_max, sigma1_min, sigma1_mean, sigma1_stdev] =
    min_max(sigma1);
42  [sigma2_max, sigma2_min, sigma2_mean, sigma2_stdev] =
    min_max(sigma2);
43
44  mfprintf(fit_res, 'm 100 org \t %3.3f \t\t %3.3f \t\t %3.3f \t\t %3.3f\n', m1_mean,
    m1_stdev, m1_min, m1_max)
45  mfprintf(fit_res, 'm 100 fit \t %3.3f \t\t %3.3f \t\t %3.3f \t\t %3.3f\n', m2_mean,
    m2_stdev, m2_min, m2_max)
46
47  mfprintf(fit_res, 's 100 org \t %3.3f \t\t %3.3f \t\t %3.3f \t\t %3.3f\n',
    sigma1_mean, sigma1_stdev, sigma1_min, sigma1_max)
48  mfprintf(fit_res, 's 100 fit \t %3.3f \t\t %3.3f \t\t %3.3f \t\t %3.3f\n',
    sigma2_mean, sigma2_stdev, sigma2_min, sigma2_max)
49
50  mclose(fit_res);
```

3.4 Walidacja programu

Każdy program niezależnie od języka w którym został napisany może zawierać błędy i warto zwalidować jego działanie. Do walidacji skryptów zaprezentowanych w poprzednim rozdziale użyto programu Origin 6.0. Jest to profesjonalny pakiet, umożliwiający pozyskiwanie, analizę i wizualizację danych pomiarowych. Obsługa programu Origin jest zbliżona do pracy z arkuszem kalkulacyjnym.
Omawiane skrypty Scilaba zapisują do pliku wygenerowane dane. Wczytano je do Orgina i dopasowano przy użyciu odpowiedniej funkcji, stosując te same parametry początkowe. Porównano wyniki. Skrypty można uznać za działające poprawnie jeśli wyniki otrzymane za ich pomocą pokrywają się z wynikami uzyskanymi przy użyciu innego narzędzia (Origin). Przeprowadzono wiele porównawczych dopasowań i zawsze wyniki otrzymane z dwóch niezależnych źródeł pokrywały się. Przykładowe porównanie dla każdego z rozważanych rozkładów prawdopodobieństwa znajduje się w tabeli 3.1. Oprócz parametrów rozkładu otrzymanych z dopasowania porównano dodatkowo χ^2.

Tabela 3.1 Walidacja skryptów za pomocą programu Origin

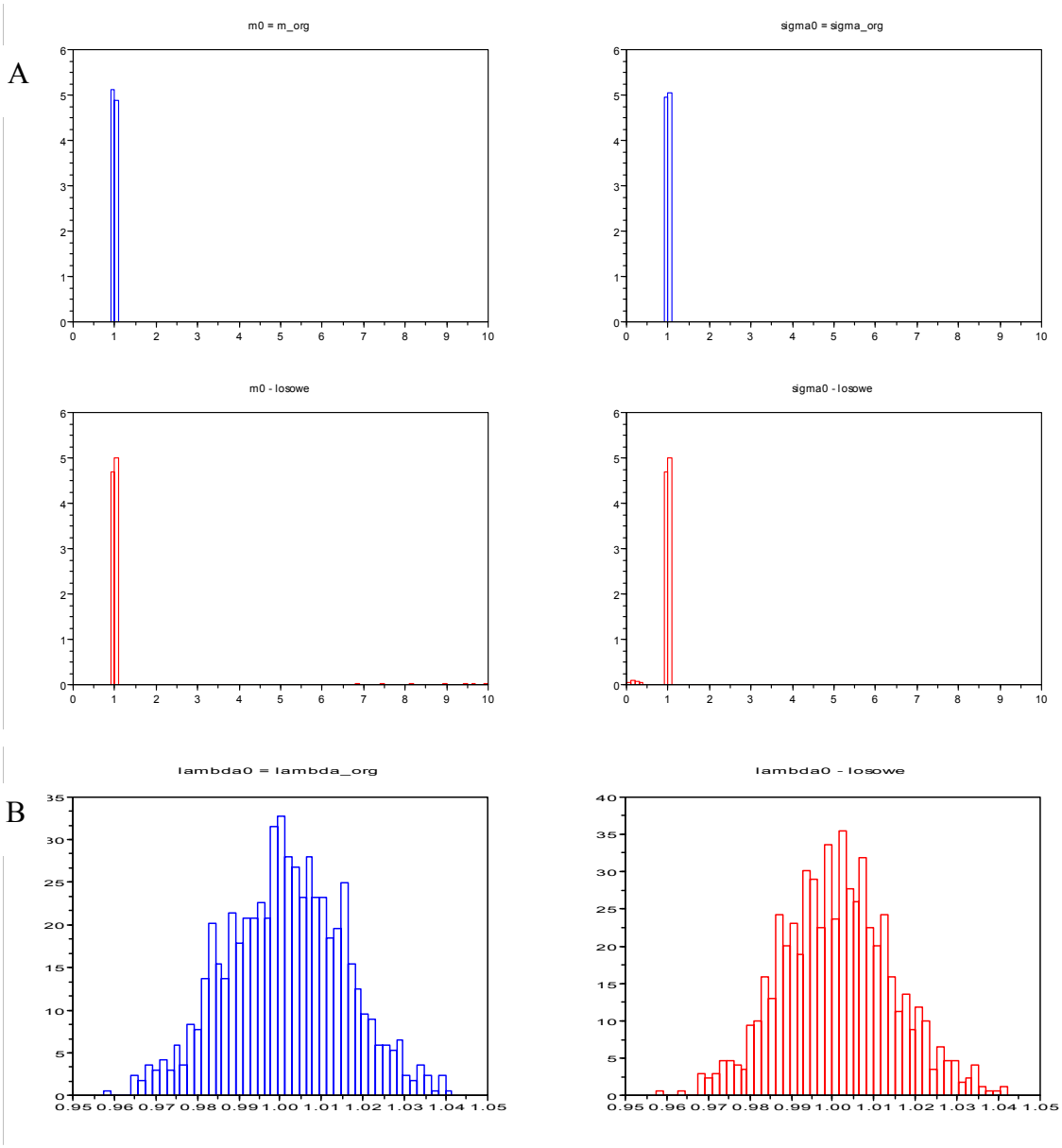
parametry	**Scilab**	**Origin**
rozkład normalny		
m	2,2854	2,2855
σ	3,2034	3,2035
χ^2	0,0002	0,0002
rozkład wykładniczy		
λ	4,7126	4,7127
χ^2	0,0307	0,0306

3.5 Wyniki działania prezentowanych skryptów

Celem tej pracy nie jest określenie możliwości wyznaczenia parametrów z rozkładów normalnego i wykładniczego w zależności od parametrów rozkładu, warunków symulacji, procesu estymacji itp, tylko przedstawienie możliwości pakietu Scilab między innymi do generowania danych, wyznaczania parametrów i wizualizacji wyników. W

rozdziale tym przedstawione zostaną przykłady wyników możliwych do uzyskania za pomocą omawianych skryptów.

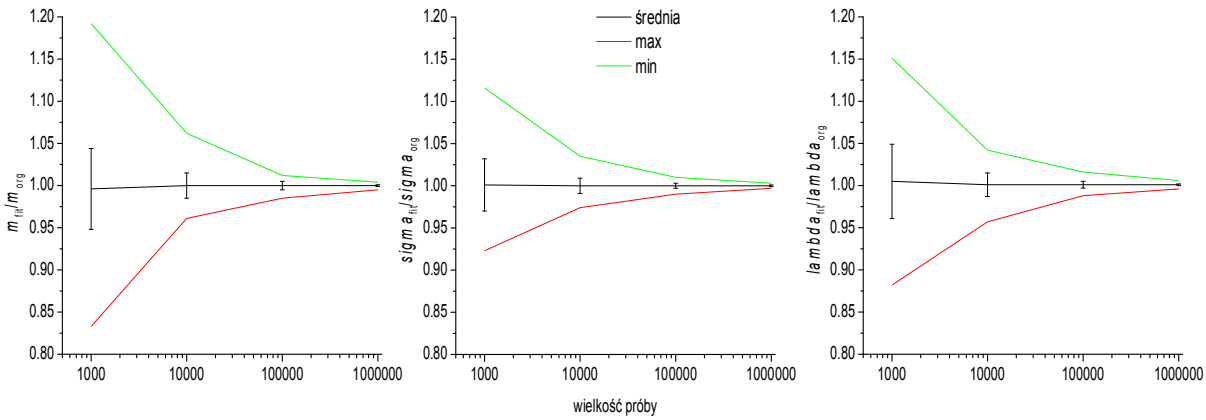
Zbadano wpływ zastosowanych punktów początkowych na otrzymane w wyniku estymacji parametry. Przeprowadzono po 10^3 symulacji, wielkość próby wynosiła 10^4. Raz stosowano punkty startowe, których wartości były równe parametrom użytym do symulacji, w drugim eksperymencie punkty startowe losowano z takiego przedziału, że w skrajnym przypadku mogły być o rząd wielkości mniejsze, lub większe od parametrów oryginalnych. Rysunek 8 przedstawia histogramy znormalizowanych parametrów (panel A dla rozkładu normalnego, B dla rozkładu wykładniczego).



Rys. 7 Histogramy znormalizowanych parametrów obrazujące wpływ punktów startowych na estymowane parametry (A – rozkład normalny; B – rozkład wykładniczy)

W przypadku rozkładu normalnego zgodnie z oczekiwaniami lepsze wyniki uzyskano przy zastosowaniu punktów startowych równych parametrom oryginalnym. Wszystkie z tysiąca wyników położone są bardzo blisko jedności. W przypadku losowych parametrów początkowych zdarzają się pojedyncze wyniki znacznie odbiegające od jedności. Jednak nawet dla punktów startowych znacznie odbiegających od wartości parametrów użytych do symulacji uzyskuje się dobre wyniki. Pokazuje to, ze dla rozważanych przypadków wyznaczane parametry nie są bardzo czułe na zastosowane punkty startowe. Parametry wyznaczane dla rozkładu wykładniczego, przy zastosowanych tutaj warunkach symulacji, okazały się nieczułe na stosowane punkty startowe.

Wykresy na rysunku 9. pokazują zależność estymowanych parametrów od wielkości wygenerowanej próby. Wyniki przedstawiono za pomocą linii reprezentujących średnia, odchylenie standardowe średniej, wartość minimalną i maksymalną z 10^3 dopasowań. Na osi Y użyto skalę logarytmiczną. Zastosowano punkty startowe zgodne z parametrami użytymi do generacji danych. Zgodnie z oczekiwaniami wyniki poprawiają się wraz ze wzrostem próby. Otrzymane wyniki nawet dla najmniejszej próby należy uznać za bardzo dobre. Pogorszenie następuje gdy wielkość próby spada do 10^2 (dane nie pokazane).



Rys. 9 Wpływ wielkości próby na estymowane parametry.

4. Podsumowanie

W rozdziale 2 zamieszczona została instrukcja ułatwiająca rozpoczęcie pracy z pakietem oraz zarysowane zostały szerokie możliwości wykorzystania Scilaba. Pakiet ten jest powszechnie używany na wyższych uczelniach, także na zajęciach dydaktycznych ze studentami. Można pokusić się o zastosowanie pakietu również na niższych stopniach edukacji co proponuje w swoim artykule „Fizyka w szkole i komputery" Jerzy Karczmarczyk [19].

W rozdziale 4 zaprezentowane zostały skrypty pokazujące zastosowanie pakietu do generacji danych, estymowania parametrów i wizualizacji wyników. Zastosowany interfejs użytkownika pozwala stworzyć narzędzie, z którego mogą korzystać osoby nie posiadające umiejętności programistycznych.

Słabością Scilaba jest między innymi niższa wydajność jego skryptów w porównaniu z programami pisanymi w językach wymagających kompilacji, stosunkowo niewielka ilość publikacji poświęconych pakietowi. Występują czasem również problemy ze stabilnością pracy z programem. Wszystkie prezentowane w tej pracy przykłady przygotowane były w wersji 4.1.2, ponieważ najnowsza piąta wersja pakietu wydaje się być niedopracowana.

Niezaprzeczalne zalety Scilaba to:

- jego niekomercyjny charakter
- dostępność pod wszystkie popularne systemy operacyjne
- stosunkowa łatwość poznania języka programowania, którym dysponuje pakiet
- bardzo szerokie, różnorodne możliwości zastosowania pakietu

Scilab nie jest pozbawiony wad, jednak powyższe cechy pakietu zachęcają do zainteresowania się tym narzędziem.

5. Bibliografia

1. http://www.scilab.org/
2. Brandt, Analiza danych, PWN, Warszawa, (1999)
3. Martyński et al., J. Chem. Phys. 122, (2005), 134507
4. Praca zbiorowa, Ćwiczenia laboratoryjne z chemii fizycznej, Wydawnictwo Naukowe UAM, Poznań, (2004)
5. Szuba, Ćwiczenia laboratoryjne z fizyki, Wydawnictwo PP, (2007)
6. http://www.mathworks.com/
7. http://www.gnu.org/software/octave/
8. Brozi, Scilab w przykładach, Nacom, Poznań, (2007)
9. Lachowicz, Matlab Scilab Maxima. Opis i przykłady zastosowań, Oficyna Wydawnicza Opole, (2005)
10. http://www.iecn.u-nancy.fr/~szulc/intrscilabdoc.pdf
11. http://matlab.pl/index.php
12. http://www.scilab.org/publications/index_publications.php?page=freebooks
13. http://www.scilab.ovh.org/ebooks.php
14. http://ftp.chg.ru/pub/math/Scilab/documentation/pdf/
15. http://www.csa.ru/~zebra/my_scilab/index.html
16. http://www.iecn.u-nancy.fr/~pincon/scilab/bench-scilab-matlab.html
17. http://www.ont.com.pl/cenniki.php
18. Czermiński i inni, Metody statystyczne dla chemików, PWN, Warszawa (1992)
19. http://www.if.uj.edu.pl/Foton/81/pdf/karczmarczuk.pdf