Laboratorium Metod Numerycznych
Postawy programowania w ***Scilab***'ie
Laboratorium 2

Korzystanie z środowiska obliczeniowego ***Scilab*** wyłącznie w oparciu o konsolę jest wyjątkowo nieefektywne. optymalnym rozwiązaniem jest zapisywanie sekwencji instrukcji do wykonania w pliku tekstowym i przekazanie jego wykonania do konsoli. Środowisko dostarcza szeregu narzędzi programistycznych wspierających pisanie programów. Zapisywanie sekwencji obliczeniowych w kryptach znacznie upraszcza proces opracowywania i późniejszego wykorzystywania algorytmu obliczeniowego. Kod skryptów można pisać dowolnym edytorze tekstowym, jednakże zdecydowanie efektywniejsze jest korzystanie z zintegrowanego z środowiskiem edytora *SciNotes*. Aby skrypt był rozpoznawany w sposób prawidłowy przez narzędzia wykonawcze ***Scilab***'a musi on być zapisany z rozszerzeniem **.sce*. Wbudowany edytor oczywiście automatycznie nadaje właściwe rozszerzenie.

Zintegrowany edytor zapewnia wsparcie przy pisaniu i testowaniu kodu:

- kolorowanie składni pozwala na szybką weryfikacje kodu,
- ułatwiony dostęp do dokumentacji funkcji i instrukcji (menu podręczne pod prawym przyciskiem myszki)[2],
- częściowe auto uzupełnianie kodu,
- numerowanie linii, ułatwiające identyfikację błędów zgłaszanych przez interpreter,
- zlecanie wykonania skryptu przez interpreter środowiska.

Opracowany algorytm w postaci skryptu i zapisany na dysku komputera może być wielokrotnie stosowany w miarę potrzeb. Podobnie jak językach ogólnego przeznaczenia można tworzyć skrypty z zestawami funkcji które można dołączać do tworzonych skryptów obliczeniowych. Tak tworzone pliki powinny być zapisywane z rozszerzeniem **.sci*. Każdy skrypt można wykonać w konsoli za pomocą instrukcji `exec()`. Za jej pomocą można także załadować zdefiniowane w pliku **.sci* funkcje.

1) Skrypt

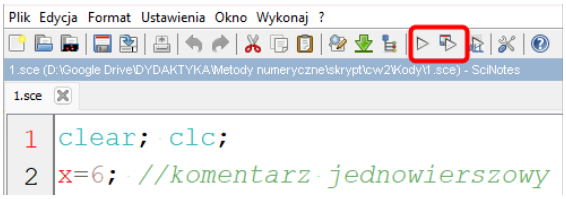
W odróżnieniu od języków programowania skrypt w środowisku ***Scilab***'a nie jest budowany w oparciu o żaden obowiązkowy szablon. Wynika to z faktu że skrypt jest zapisem ciągu instrukcji przekazywanych sekwencyjnie do wykonania przez interpreter środowiska zintegrowany w konsoli. Jednakże ze względów praktycznych można każdy skrypt tworzyć pewnym stałym układzie składającym się z trzech zasadniczych części:

1. czyszczenie pamięci (`clear`), konsoli (`clc`), okna graficznego (`clf`) i ładowanie plików z funkcjami (`exec()`, opcjonalne),
2. kod funkcji,
3. kod właściwy skryptu.

Przykładowy skrypt realizujący proste zadanie obliczeniowe na zmiennych pokazano poniżej. Zwrócić należy na dwa sposoby dodawania komentarzy w kodzie skryptu, komentarz jedno i wielowierszowe. Stosowanie komentarzy w trakcie pisania kodu skryptu ułatwi analizę kodu innym jego użytkownikom, jak również pozwoli na ułatwiona jego reedycje po czasie gdy autor może nie pamiętać szczegółów. Komentarze pozwalają także w wygodny sposób wyłączyć określone fragmenty kodu z wykonania ich przez interpreter w trakcie tworzenia kodu skryptu.

```
clear; clc;
x=6; //komentarz jednowierszowy
y=24;
/* Komentarz
wielowierszowy
z=0; */
z=y/x; /* efekt działania kodu zakończony
          średnikiem nie będzie wyświetlony
          w konsoli */
z //efekt działania linii będzie widoczny w konsoli
```

Zapisany powyżej skrypt można wykonać z poziomu edytora *SciNotes* za pomocą przycisku *Wykonaj* lub *Zapisz i Wykonaj* jak pokazana na Rys.1 lub za pomocą polecenia `exec()` w konsoli. Należy tu zwrócić uwagę na dwie różnice jakie można zaobserwować. Wykonanie skryptu z poziomu edytora nie wywoła żadnego efektu w konsoli. W przypadku drugim wyświetlone zostaną wszystkie zlecone do wykonania polecenia a wyniki ich działania będą widoczne tylko w liniach



Rys. 1: Przyciski wykonania skryptu z poziomu edytora SciNotes

Tab. 1: Tryby działania instrukcji exec()

atrybut	efekt działania	konsola
-1	efekt działania skryptu nie jest wyświetlany	`-->`

0	wyświetlany jest efekt działania linii nie zakończonych średnikiem	`z =`
		`4.`
		`-->`

1 lub brak	wyświetlane są instrukcje i efekt działania linii kodu nie zakończonych średnikiem	`--> x=6;`
		`--> y=24;`
		`--> z=y/x;`
		`--> z`
		`z =`
		`4.`

7	efekt podobny do tego dla atrybutu 1, ale wykonywanie kolejnych linii kodu realizowane jest po naciśnięciu klawisza ENTER	

które nie są zakończone średnikiem. Sposób wykonania skryptu w konsoli przez polecenie `exec()` może być sterowane drugim atrybutem instrukcji, jego wartości i efekt działania zestawiono w tabeli 1.

Wykonując skrypt z poziomu konsoli należy się upewnić czy aktywny katalog roboczy jest ten w którym znajduje się skrypt do wykonania. Identyfikacje folderu roboczego można zrealizować za pomocą polecenia `pwd` a następnie jego zmianę za pomocą instrukcji `cd` (składnia analogiczna do znanej z systemu *ms dos/windows*). Wygodniejsze jednak będzie przełączeni katalogu roboczego za pośrednictwem okna przeglądarki plików (na lewo od konsoli).

2) instrukcje interfejsu

Pisząc skrypt zazwyczaj konieczne będzie opracowanie interfejsu komunikacji z użytkownikiem:

- pobranie informacji (danych) Najczęściej stosowana będzie instrukcja `input()` wyświetlająca na ekranie konsoli ciąg tekstowy poprzedzający znak zachęty do wprowadzenia danych i przypisując podane przez użytkownika informacje do wskazanej zmiennej, jak pokazano poniżej.

  ```
  x=input("Podaj liczbę=");
  ```

- wyświetlenie wyników obliczeń W zależności od specyfiki zadania realizowanego przez skrypt wyniki mogą byc zwracane na trzy sposoby:

– dane liczbowe w konsoli,

– graficznie w postaci wykresu,

– zapis danych do pliku.

Pierwszy z nich realizować można za pomocą omówionych w instrukcji 1 funkcji `disp()` i `mprintf()`, jak pokazano przykładowym skrypcie pokazany poniżej. Prezentacja wyników w sposób graficzny zostanie omówiona w kolejnej instrukcji, a trzeci sposób zostanie przybliżony pod koniec tej instrukcji.

```
clear; clc;
x=input("Określ rozmiar macierzy=");
Z=eye(x,x);
disp("Podano rozmiar n="+string(x));
disp("Macierz Z:",Z);
mprintf("Wyznacznik macierzy det(Z)=%f",det(Z));
```

Efekt działania powyższego kodu pokazano poniżej.

```
Określ rozmiar macierzy=4
  "Podano rozmiar n=4"
  "Macierz Z:"
   1.   0.   0.   0.
   0.   1.   0.   0.
   0.   0.   1.   0.
   0.   0.   0.   1.
Wyznacznik macierzy det(Z)=1.000000
```

W pierwszej konstrukcji instrukcji `disp()` zbudowano jeden ciąg tekstowy, natomiast w drugim każdy z argumentów rozdzielonych przecinkiem jest wykonywany i wyświetlany niezależnie.

3) Decyzje

W kodzie ***Scilab***'a można zaprogramować warunkowe wykonanie kodu za pomocą instrukcji warunkowej lub wyboru. Decyzja podejmowana jest na podstawie zaprogramowanego warunku logicznego. Test logiczny budowany jest w oparciu o podstawowe operatory relacji. Możliwe jest także budowanie złożonych warunków logicznych z wykorzystaniem operatorów. Operatory stosowane przy konstruowaniu warunków logicznych zestawiono w tabeli w tabeli 2.

Podstawowa konstrukcją testu logicznego jest instrukcja warunkowa `if warunek then kod end`. Konstrukcja ta wykonuje kod umieszczony pomiędzy łowami kluczowymi `then` i `end` gdy warunek logiczny zapisany po `if` zwróci wartość `TRUE`.

Tab. 2: Operatory relacji i logiczne

Operator	Opis
Operatory relacji	
==	równości
= lub <>	różne
>	większy niż
<	mniejszy niż
<=	mniejszy lub równy
>=	większy lub równy
Operatory logiczne	
&	iloczyn logiczny (AND)
|	suma logiczna (OR)
~	negacja (NOT)

W przykładowym kodzie pokazanym poniżej testowana jest wartość liczby wprowadzonej przez użytkownika i jeżeli spełniony jest warunek (liczba większa od zera) wykonywany jest kod generujący kwadratową macierz losową o zadanym przez liczbę rozmiarze.

```
clear; clc;
x=input("Określ rozmiar macierzy=");
if x>0 then
    Z=rand(x,x);
    disp("Macierz Z:",Z);
end
```

Jeżeli podana zostanie liczba dodania wygenerowana zostanie macierz losowa, jak pokazano poniżej, w przeciwnym wypadku nic się nie stanie.

```
Określ rozmiar macierzy=5
  "Macierz Z:"
   0.2113249   0.6283918   0.5608486   0.2320748   0.3076091
   0.7560439   0.8497452   0.6623569   0.2312237   0.9329616
   0.0002211   0.685731    0.7263507   0.2164633   0.2146008
   0.3303271   0.8782165   0.1985144   0.8833888   0.312642
   0.6653811   0.068374    0.5442573   0.6525135   0.3616361
```

Podobnie jak językach programowania możliwe jest zaprogramowania także reakcji instrukcji `if` na niespełnieni warunku testowego. Realizuje się to poprzez rozbudowanie konstrukcji o słowo kluczowe `else`, jak pokazano na poniższym przykładzie. Po kodzie realizowanym w przypadku spełnienia warunku należy umieścić instrukcje `else`, a po niej kod realizujący zadania w przypadku niespełnienia warunku instrukcji warunkowej.

```
clear; clc;
x=input("Określ rozmiar macierzy=");
if x>0 then
    Z=rand(x,x);
    disp("Macierz Z:",Z);
else
    disp("Podano liczbę mniejszą od zera.")
    disp("Nie można utworzyć takiej macierzy.")
end
```

Instrukcje warunkową `if` można rozbudować o kolejne stopie, tak jak i w innych języka programowania. W środowisku ***Scilab*** kolejny stopień dodaje się za pomocą słowa kluczowego `elseif warunek then`. Budowana kaskada może składać się z kilku stopni realizujących różne zadania w zależności od spełnienia zapisanych warunków. Należy pamiętać że kaskada warunkowa może nie wykonać żadnego z z niej kodów (gdy nie ma w niej stopnia `else`), lub jeden z nich (jeżeli stopień `else` został do kaskady dołączony). Poniżej pokazano przykładowy kod skryptu z kaskadą warunkową.

```
clear; clc;
x=input("Określ rozmiar macierzy=");
if x>0 & x<7 then
  Z=rand(x,x);
  disp("Macierz Z:",Z);
elseif x>7 then
  disp("Podano za duży rozmiar macierzy,");
  disp("wygenerowana zostanie macierz w dopuszczalnym rozmiarze n=6.");
  Z=rand(6,6);
  disp("Macierz Z:",Z);
else
  disp("Podano liczbę mniejszą od zera.")
  disp("Nie można utworzyć takiej macierzy.")
end
```

Efekt działania kodu pokazano poniżej.

```
Określ rozmiar macierzy=12
  "Podano za duży rozmiar macierzy,"
  "wygenerowana zostanie macierz w dopuszczalnym rozmiarze n=6."
  "Macierz Z:"
   0.4204123   0.251896    0.0417362   0.0540932   0.8387927   0.4311733
   0.4277572   0.4391129   0.3438272   0.9190207   0.4343749   0.6145385
   0.3184586   0.0759304   0.1970167   0.4603516   0.7767876   0.9258962
   0.5761894   0.255938    0.2122899   0.2992685   0.1395318   0.0993817
   0.4254902   0.0670617   0.3140399   0.0029166   0.1150637   0.4280579
   0.9761982   0.7651132   0.7821625   0.8993471   0.535542    0.9431831
```

Słowo kluczowe `then` w składni instrukcji warunkowej nie jest wymagane i może być pominięte, jednakże jego stosowanie poprawia czytelność kodu i zakres kodu warunku logicznego jest wyraźnie określony. Z tego względu zaleca się stosowanie pełnej składni instrukcji `if`.

4) Instrukcja wyboru

W wielu sytuacja korzystanie z rozbudowanej kaskady `if elseif` możne być mało efektywne, złożone konstrukcyjni i nieczytelne. Jeżeli takiej kaskadzie rozpatrywana jest wartość zmiennej jako kryterium decyzyjne, kozystej jest zastosować instrukcje wyboru analogiczną do `switch case` znana z języka `C\C++`. W środowisku ***Scilab***'a jest dostępna konstrukcja analogiczna oparta o instrukcje `select case`. Konstrukcja składa się z trzech części:

- przełącznik `select` wskazujący na zmienna sterującą,
- listę bloków z etykietą `case` wraz z wartością przechwytującą wykonanie umieszczonego po niej kodu dla wskazanej wartości parametru sterującego,
- blok `else` (domyślny) umieszczony w nim kod jest wykonywany gdy parametr sterujący nie przyjął wartości zgodnej z którymkolwiek z zestawionych w liniach `case`.

Należy mieć świadomość ograniczenia instrukcji wyboru pozwalającej realizować przełączanie pomiędzy zaprogramowanymi blokami kodu na podstawie ściśle określonej wartości, czyli tylko dla liczb całkowitych i znaków (ciągów znakowych). Poniżej zapisano przykładowy kod zrealizowany w oparciu o instrukcje wyboru `select`.

```
clear; clc;
disp("Program oblicza A. sin(x) lub B. cos(x) liczby.");
x=input("Podaj wartość liczby x=");
w=input("Wybierz działanie (A lub B):","string");
switch w
case 'A' then
    disp("Sin("+string(x)+")="+string(sind(x)));
case 'B' then
    disp("Cos("+string(x)+")="+string(cosd(x)));
else
    disp("Niewłaściwy wybór!");
end
```

Można zwrócić uwagę na użycie apostrofów i cudzysłowów w powyższym kodzie. Ich użycie jest wymienne,jednakże ze względu na czytelność kodu i poprawność składniowa zaleca się stosowanie cudzysłowów dla ciągów tekstowych a apostrofów dla znaków, analogicznie jak to było przyjęte w standardzie języka `C\C++`.

5) Pętle

Możliwość powtarzania kodu jest kolejną cecha języków programowania zaimplementowaną w środowisku ***Scilab***'a. Dostępne są dwie podstawowe konstrukcje pętli: iteracyjna i warunkowa.

5.1 Pelta iteracyjna for

Pętla iteracyjna realizuje określoną liczbę powtórzeń powiązanego z nią bloku kodu. Liczba powtórzeń jest zależna od rozmiaru wektora danych powiązanego z zmienną iteracyjna pętli. Innymi słowy pętla realizowana jest poprzez przełączanie zmiennej sterującej po elementach wektora. Jest to rozwiązanie analogiczne do stosowanego w języku `python` i jednej z postaci instrukcji `for` w języku `C++`.

Wektor zdefiniowany na dowolny z dostępnych w ***Scilab***'ie sposobów jest w pętli przypisywany do zmiennej iteracyjnej (o dowolnej nazwie). Przy każdej iteracji pętli zmienna przyjmuje kolejną wartość z wektora. Po przetworzeniu wszystkich elementów wektora pętla kończy działanie i jest realizowany kod po instrukcji `end` kończącej blok pętli. Poniżej pokazano przykład z implementacją rożnych deklaracji pętli iteracyjnej.

```
clear; clc;
A=[2,4,6,8];
j=0;
for i=A
    j=j+1;
    disp("A["+string(j)+"]="+string(i));
end
for i=1:5
    disp("B["+string(i)+"]="+string(i));
end
for i=5:-1:1
    disp("C["+string(6-i)+"]="+string(i));
end
j=0;
for i=linspace(6,10,5) //Może być także logspace()
    j=j+1;
    disp("D["+string(j)+"]="+string(i));
end
j=0;
for i=["jeden","dwa","trzy","cztery","pięć"]
    j=j+1;
    disp("E["+string(j)+"]="+i);
end
```

5.2 Pętla warunkowa while

Pętla warunkowa jest pewnego rodzaju hybryda pętli iteracyjnej i instrukcji warunkowej. Realizuje on powtórzenie bloku kodu z nią sprzężonego tak długo jak spełniony jest zdefiniowany w niej warunek logiczny. W środowisku ***Scilab***'a dostępne są trzy równoważne formy deklaracji pętli warunkowej, jak pokazano poniżej.

```
while warunek then
   kod;
end
```

```
while warunek do
   kod;
end
```

```
while warunek,
   kod;
end
```

Zastosowanie pętli warunkowej pokazano na poniższym przykładzie, w którym powtarzane jest losowanie liczby całkowitej przedziale 0-2·x do czasu wylosowania liczby równej x. Po zakończeniu pętli wyświetlana jest informacja o liczbie wykonanych powtórzeń kodu.

```
clear; clc;
x = input("Podaj testową liczbę dodatnią >0 (int) x=");
i=0;
n=0;
while i~=x do
    n=n+1;
    i = round((2*x)*rand());
end
disp("Liczbę "+string(x)+" wylosowano po "+string(n)+" losowaniach");
```

Efekt działania powyższego kodu pokazano poniżej.

```
Podaj testową liczbę dodatnią >0 (int) x=7
  "Liczbę 7 wylosowano po 9 losowaniach"
```

5.3 Instrukcje przerwania pętli

W środowisku Scilab'a możliwe jest zaprogramowanie przerwania działania pętli. Dostępne są instrukcje znane z innych języków programowania (jak `C\C++`, `python`): `continue` i `break`. Pierwsza z nich przerywa aktualna iterację i przechodzi o instrukcji sterującej w celu sprawdzenia warunku powtórzenia peli i wykonania kolejnej iteracji. Druga powoduje zakończenie pętli i przejście do wykonywania kodu zapisanego po pętli. Instrukcje te zazwyczaj działają w połączeniu z instrukcją warunkową `if`. Dzięki temu możliwe jest sterowanie ich aktywacją.

6) Funkcje

Kod skryptu pisanego w ***Scilab***'ie podobnie jak w innych języka może zostać podzielony pod kontem zadań realizowanych przez poszczególne jego fragmenty. Kod

takich zadań można przenieść do funkcji. Jest to szczególnie istotne gdy takie zadania są wykonywane w kilku miejscach skryptu, dzięki temu można w sposób znaczący poprawić czytelność i efektywność kodu. Drugą zaletą wydzielenia kodu do funkcji jest możliwość ponownego wykorzystania kodu w kolejnych skryptach. W tym przypadku warto wydzielić takie funkcje do osobnego pliku (`*.sci`) i wczytać taki plik na początku skryptu.

Funkcje należy deklarować na początku kodu, po instrukcjach czyszczących ale prze kodem właściwym skryptu. Funkcje w ***Scilab***'ie można zdefiniować na da sposoby:

1. pełny - w bloku deklaracyjnym `function ... endfunction` - podstawowy sposób deklaracji funkcji, składa się z dwóch części: *definicji* i *kodu*. W pierwszej określa się liczbę zwracanych wartości, nazwę funkcji i listę atrybutów wywołania. Funkcje w ***Scilab***'ie mogą zwracać zarówno pojedyncze wartości jak i tablice. Poniżej zapisano przykład deklaracji i wykorzystania funkcji realizującej zadanie z poprzedniego przykładu.

```
clear; clc;
function n = test(x)
  i=0;
  n=0;
  while i~=x do
    n=n+1;
    i = round((2*x)*rand());
  end
endfunction

x = input("Podaj testową liczbę dodatnią >0 (int) x=");
n=test(x);
disp("Liczbę "+string(x)+" wylosowano po "+string(n)+" losowaniach");
```

   Funkcja w ***Scilab***'ie może zwraca więcej jak jedna wartość. Ze względu na właściwości zmiennych zwracane wartości moa być dowolnego typu i rodzaju, także tablicowego. Wywołanie takiej funkcji wymaga przypisania jej o tablicy zawierającej listę zmiennych przyjmujących wartości zwracanych przez funkcje. Przykład skryptu z funkcją zwracającą dwie wartości pokazano poniżej.

```
clear; clc;
function [A,B] = test(x)
  A=[0];B=[0]
  for i=1:x
    A(i)=round(90*rand());
    B(i)=sind(A(i));
  end
```

```
endfunction

x = input("Podaj liczbe losowanych liczb x=");
[L,S]=test(x);
for i=1:x
  disp("sin("+string(L(i))+")="+string(S(i)));
end
```

Ponowne wykorzystanie kodu funkcji wygodnie jest realizować poprzez gromadzenie funkcji tematycznie w plikach biblioteki (`*.sci`) i wczytywać je na początku skryptu w którym mają być zastosowane, jak pokazano w poniższym przykładzie. Należy pamiętać że konieczne jest wskazanie dokładnej pozycji pliku biblioteki w strukturze katalogowej, albo ustawienie katalogu roboczego ***Scilab***'a na folder z wczytywanym plikiem.

```
//plik biblioteki fun.sci
  function [A,B] = test(x)
    A=[0];B=[0]
    for i=1:x
      A(i)=round(90*rand());
      B(i)=sind(A(i));
    end
  endfunction
  function drukuj(L,S)
    for i=1:x
      disp("sin("+string(L(i))+")="+string(S(i)));
    end
  endfunction
\\plik skryptu skrypt.sce
  clear; clc;
  exec lib\fun.sci;
  x = input("Podaj liczbe losowanych liczb x=");
  [L,S]=test(x);
  drukuj(L,S);
```

W przykładzie ustawiono katalog roboczy na folder z plikiem skryptu, a bibliotekę umieszczono w podkatalogu *lib*.

2. skrócony - z pomocą instrukcji definiującej `deff()` - pod względem efektu jest identyczna o omówionej powyżej konstrukcji `function`, ale przeznaczona do deklarowania funkcji o prostej i krótkiej składni. Kod deklarowanej funkcji jest atrybutem instrukcji `deff`. Ma dwa atrybuty rozdzielone przecinkiem:

- deklarację wywołania funkcji - analogiczna do tej z `function [zmienne]=nazwa(atr` Deklaracja jest zamknięta w apostrofach.
- kod funkcji w nawiasie kwadratowym i apostrofach

Przykład kodu skryptu z deklaracją funkcji za pomocą instrukcji `deff()` pokazano poniżej.

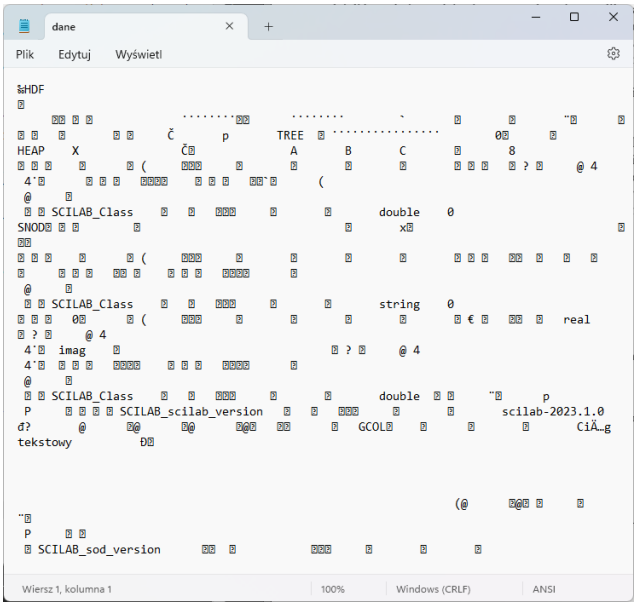
```
clear; clc;
deff('[a,b]=test(x)',['a=x^2; b=sind(a);']);
x = input("Podaj wartość tesową x=");
[a,b]=test(x);
disp(a,b);
```

7) Zapis i odczyt danych z pliku

Dane przetwarzane za pomocą zmiennych przechowywane są w pamięci, są więc to informacje ulotne. Przetwarzane w programie informacje można w wygodny i szybki sposób zapisywać i odczytywać z pliku. ***Scilab*** pozwala na zapis i odczyty danych w rybie binarnym. Możliwy jest także zapis danych do pliku tekstowego w formie czytelnej dla człowieka.

Zapis w trybie binarnym realizowany jest przez funkcje `save()`. Atrybuty można podzielić na dwie części, pierwsza wskazuje na plik na dysku, a druga jest lista zmiennych do zapisania. W przypadku pliku należy pamiętać że punktem odniesienia jest aktualny katalog roboczy ***Scilab***'a. Stąd tez albo należny właściwie zaadresować plik (względnie lub bezwzględnie) albo ustawić oczekiwany katalog roboczy.

Odczyt trybie binarnym realizowany jest przez funkcje load(). Ma ona analogiczną składnie co funkcja zapisu, jednakże lista zmiennych atrybutem opcjonalnym. Zapisywane funkcją save() są nie tylko dane ale i nazwy zmiennych które zostały o funkcji przekazane. Powoduje to że przy odczycie można pominąć listę zmiennych, one i tak zostaną odtworzone, wraz z nazwami. Możliwe jest także selektywne odczytywanie danych, to znaczy należy podać nazwę zmiennej do odczytania i tylko ona zostanie odczytana. Ta właściwość funkcji load() powoduje że nie można podać na liście dowolnych nazw zmiennych, a tylko takie które zostały zapisane. Poniżej zapisano przykład zapisu



Rys. 2: Plik binarny z danymi z przykładu

i odczytu danych w rybie binarnym, a na Rys.2 pokazano podgląd utworzonego pliku binarnego.

```
clear;clc;
A=[1,2,3,4,5];
B="Ciąg tekstowy";
C=12+3*%i;
save("dane.dat",'A','B','C');
clear;
load("dane.dat"); //odczytanie wszystkich danych z pliku
disp(A,B,C);
clear;
load("dane.dat",'B'); //odczytanie tylko zmiennej B
disp(B);
```

Zapis zawartości zmiennej do pliku tekstowego realizowany jet przez funkcje `print()`. Sam zapis jest odpowiednikiem wyświetlania danych w konsoli przez funkcję `disp('nazwa',` W przypadku zapisu do pliku tekstowego podaje się wyłącznie nazwę pliku, rozszerzeni jest nadawane automatycznie. Zapisywana jest wyłącznie zawartość zmiennych.

Większość urządzeń pomiarowych zapewnia eksport danych do pliku w formacje `*.csv`. ***Scilab*** pozwala na import danych za pośrednictwem dedykowanej funkcji `csvRead()`. Prawidłowe skonfigurowanie atrybutów funkcji wymaga bezpośredniej analizy budowy pliku `*.csv` i określenia pozycji danych które maja zostać zaimportowane. Należy zwrócić uwagę na sposób zapisu znaku oddzielającego cześć ułamkową liczby rzeczywistej, ***Scilab*** stosuje znak kropki. Konieczna może być konwersja znaku o kropki przed odczytem danych. W Podstawowej formie funkcja wymaga podania nazwy pliku `*.csv` i znaku (znaków) rozdzielających kolumny danych w pliku, jak pokazano w poniższym przykładzie.

```
X=csvRead("dane.csv",';')
```

Odczytane zostaną wszystkie dane z pliku i zapisane w tablicy X, jak pokazano poniżej.

```
X =
  Nan Nan
  0.01 0.0000005
  0.02 0.000016
  0.03 0.0001215
  0.04 0.000512
  0.05 0.0015625
  0.06 0.003888
  0.07 0.0084035
  0.08 0.016384
  130.09 0.0295245
  0.1 0.05
```

Pozycje Nan oznaczają niekompatybilne dane '*Not a number*', zazwyczaj stanowiące opis poszczególnych kolumn w pliku. Możliwe jest odczytanie selektywne z pliku ze wskazaniem kolumn i wierszy do odczytania, jak pokazano poniżej.

```
X=csvRead("Zeszyt1.csv",';',[],[],[],[],[2 1 11 2])
```

W składni tej istotna jest liczba pustych nawiasów kwadratowych odpowiadających nieużytym atrybutom funkcji. Istotny jest ostatni nawias w którym określono zakres importowanych danych:

- 2 - numer wiersz od którego rozpoczyna się import danych
- 1 - numer kolumny od której rozpoczyna się import danych
- 11 - numer wiersz na którym kończy się import danych
- 2 - numer kolumny na której kończy się import danych

W tak zaprogramowanym imporcie danych pominięty zostanie pierwszy wiersz z danymi nieliczbowymi i w tablicy X zostaną zapisane wyłącznie dane liczbowe.

Dane przetworzone w skrypcie ***Scilab***'a mogą zostać zapisane w standardzie pliku `*.csv`. Zadanie to realizuje funkcja `csvWrite()` która zapisuje tablicę danych do wskazanego pliku `*.csv`. Tablica może być przekazana jako pojedyncza zmienna lub jako zestaw wektorów, jak pokazano w przykładzie poniżej. Istotne są także atrybutu trzeci, określający separator kolumn i czwarty którym można określić separator dziesiętny. Dobór tych parametrów jest istotny pod katem wyboru aplikacji docelowej dla otwarcia zapisanego pliku danych. W przypadku Excela na separator kolumn można wybrać znak średnika, a na separator dziesiętny należy wybrać znak przecinka. Jeżeli żaden z tych dwóch atrybutów nie zostanie podany, w trybie domyślnym ustawione zostaną dla separatora kolumnowego przecinek a dziesiętnego kropka.

```
clear;clc;
a=linspace(1,100,100);
b=logspace(0,2,100);
B=[a;b];
csvWrite([a' b'],"noweDane1.csv",';',',');
csvWrite(B','"noweDane2.csv",';',',');
```

Zastosowana w przykładzie transpozycja macierzy pozwoliła na zapisanie danych w układzie kolumnowym.

Możliwy jest także import danych z pliku arkusza kalkulacyjnego Excel, co zostanie omówione w jednej z kolejnych instrukcji.

8) Wykresy

Ostatnim z omawianych w niniejszej instrukcji, ale chyba najważniejszym sposobem prezentacji wyników obliczeń jest graficzna prezentacja danych w postaci wykresów. ***Scilab*** pozwala na wkreślanie danych za pomocą wielu typów wykresów. Jednakże do celów inżynierskich najistotniejsze są trzy z nich:

- wykres punkowy (XY) z interpolacją - przebieg zmian analizowanej wielkości
- wykres powierzchniowy ($3D$) - mapa rozkładu analizowanej wielkości
- histogram - analiza rozkładu empirycznego badanej wielkości Każdy generowany wykres powinien poza wybrana formą prezentacji danych zapewniać:
- informacje o prezentowanych danych za pomocą "*czytelnego*"tytułu wykresu
- możliwość do precyzyjnego odniesienia się do prezentowanych danych za pomocą właściwie wyskalowanych i opisanych osi
- przy prezentacji kilku rożnych charakterystyk możliwość jednoznacznej ich identyfikacji za pomocą jednoznacznego oznaczenia poszczególnych przebiegów i identyfikacji ich za pomocą precyzyjnych opisów umieszczonych w legendzie

Każdy skrypt ***Scilab***'a operujący na oknie graficznym poza czyszczeniem konsoli i pamięci na początku powinien także czyścić lub kasować okno graficzne.

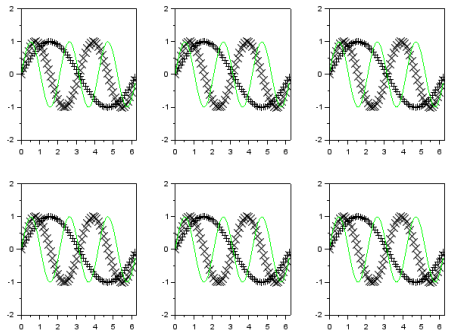
- clf; - czyszczenie zawartości otwartych okien graficznych
- close(); - kasowanie otwartych okien graficznych

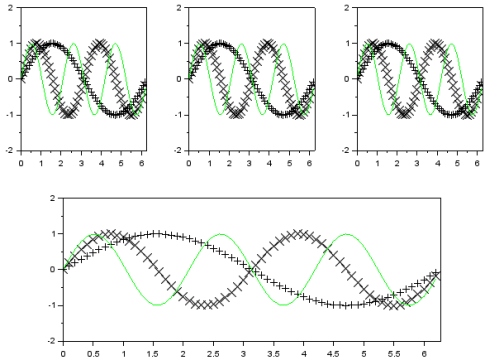
W wielu sytuacja może pojawić się konieczność zaprezentowania danych uzyskanych z symulacji za pomocą kilku wykresów. ***Scilab*** pozwala na umieszczenie w zorganizowanej formie określoną liczbę wykresów w jednym oknie graficznym. Drugą opcja jest otwarcie niezależnego okna graficznego dla każdego z tworzonych wykresów.

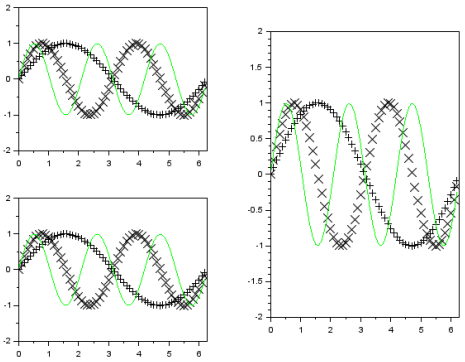
Współdzielenie pojedynczego okna graficznego przez określona liczbę wykresów realizowane jest to za pomocą polecenia `subplot()`. Funkcja ta dzieli obszar okna graficznego na macierz pól graficznych określonych przez liczbę wierszy i kolumn. Definiują to dwa pierwsze atrybuty funkcji, natomiast trzeci aktywuje wybrane pole. Każde pole ma swój numer, począwszy od 1 dla lewego, górnego pola macierzy. Poniżej, w tabeli 3, zestawiono kilka przykładowych zorganizowanych przez instrukcję `subplot()` okien graficznych. Należy zwrócić uwagę na nakładanie się na siebie różnych rozkładów.

Tab. 3: Przykłady organizacji okna graficznego funkcją subplot().

Kod	Okno graficzne

`clear;clc;`
`close();`
`for i=1:6`
`  subplot(2,3,i);`
`  plot2d();`
`end`
	

`clear;clc;`
`close();`
`for i=1:3`
`subplot(2,3,i);`
`plot2d();`
`end`
`subplot(2,1,2);`
`plot2d();`
	

`clear;clc;`
`close();`
`for i=1:2`
`subplot(2,2,2*i-1);`
`plot2d();`
`end`
`subplot(1,2,2);`
`plot2d();`
	

Zarządzanie niezależnymi oknami graficznymi realizowane jest przez funkcję scf(). W podejściu tym możemy wyszczególnić trzy typy czynności:

- Utworzenie nowego okna graficznego, automatycznie okno ustawiane jest jako aktywne –> `nazwa = scf(ID);`
- Aktywowanie wskazanie okna (wcześniej utworzonego) –> `scf(nazwa);` lub `scf(ID);`
- Skasowanie wybranego okna graficznego –> `close(nazwa);` lub `close(ID);`

Poniżej zapisano przykład obrazujący omówione powyżej instrukcję. Skrypt tworzy dwa okna graficzne umieszczając w nich kolejno wykresy. Następnie okna są kasowane, a na koniec okno o $ID = 2$ jest ponownie otwierane i umieszczany jest nim wykres.

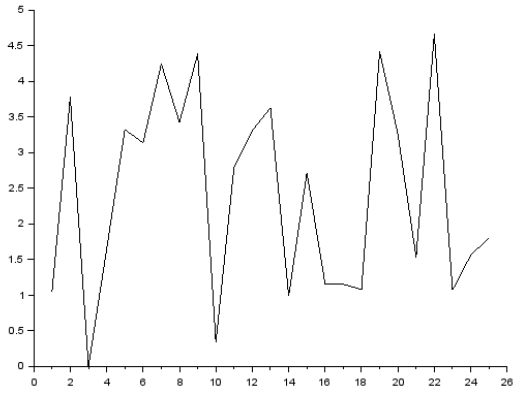
```
clear;clc;close();
okno_1=scf(1);
okno_2=scf(2);
  plot2d();
scf(okno_1);
  subplot(1,2,1); plot2d();
  subplot(1,2,2); plot2d();
close(okno_2); close(1);
scf(2); plot2d();
```

8.1 Wykres punkowy

Jest to podstawowa forma wykresów generowanych dla prezentacji danych pomiarowych czy symulacyjnych. Pozwala na wykreślenie danych określonych przez zestaw punktów opisanych w przestrzenie *2D*. Wykres generowany jest za pomocą funkcji `plot2d()`, opis osi i tytuł wykresu za pomocą `xtitle()`, natomiast legenda za pomocą `legend()` lub `legends()`.

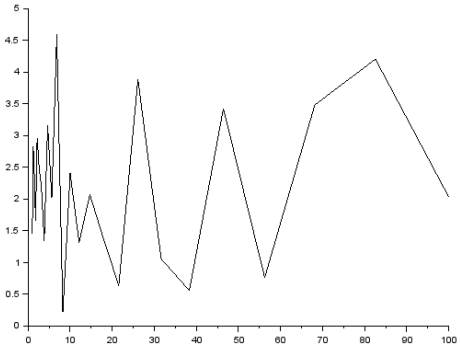
Podstawowa forma wywołania funkcji `plot2d()` wymaga przekazania jako atrybutu wektora z danymi do wykreślenia. W takim przypadku dane względem osi *Ox* zostaną wykreślone względem pozycji tablicy, jak pokazano poniżej.

```
clear;clc;close();
y=5*rand(1,25);
plot2d(y);
```



Dodając do atrybutów wywołania funkcji `plot2d()` wektor danych dla współrzędnych na osi Ox uzyskuje się wyskalowanie obu osi wykresu do danych wejściowych opisujących rozkład punktów w przestrzeni Oxy zdefiniowanych przez wektory danych wejściowych, jak pokazano poniżej.

```
clear;clc;close();
x=logspace(0,2,25);
y=5*rand(1,25);
plot2d(x,y);
```

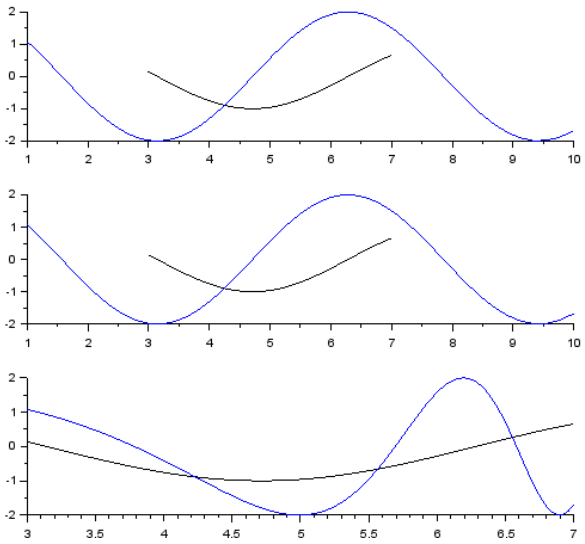


Funkcja `plot2d()` umożliwia umieszczanie na wspólnym wykresie kilka charakterystyk. Można to zrealizować na trzy sposoby:

- niezależne wywołanie funkcji `plot2d()` do wykreślenia każdego z wykresów. Skalowanie osi dobierane jest automatycznie do zakresu wykreślanych danych
- pojedyncze wywołanie z wspólnym wektorem opisującym dane dla osi Ox, osie wyskalowane dla wszystkich charakterystyk
- pojedyncze wywołanie z niezależnymi zestawami danych dla osi Ox i Oy, drugi zestaw danych decyduje o skalowaniu osi, konieczne jest dodanie osi pomocniczych pozostałych charakterystyk - rozwiązanie niezalecane, niepraktyczne

Poniżej zapisano przykład prezentujący omówione powyżej wywołania funkcji `plot2d()`.

```
clear;clc;close;
x1=linspace(3,7,100);
x2=logspace(0,1,100);
y1=sin(x1);
y2=2*cos(x2);
subplot(3,1,1);
plot2d(x1,y1,1);
plot2d(x2,y2,2);
subplot(3,1,2);
plot2d([x1' x2'],[y1' y2']);
subplot(3,1,3);
plot2d(x1,[y1' y2']);
```



Zastosowane w przykładzie apostrofy przy wektorach danych są wymagane do obrócenia wektora wierszowego na kolumnowy aby zapewnić zgodność z wektorem danych osi Ox.

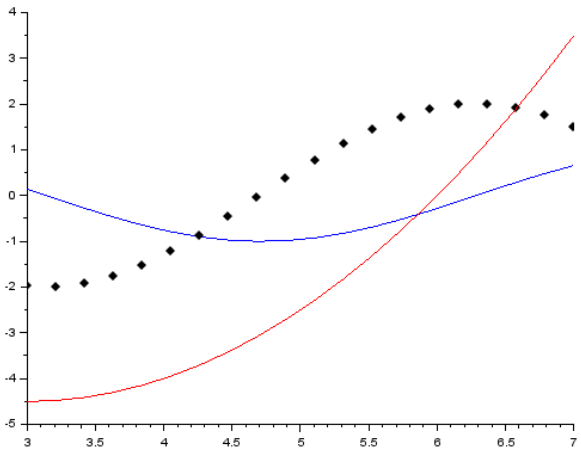
Funkcja `plot2d` posiada jeszcze kilka opcjonalnych atrybutów pozwalających na:

- ustawienie koloru lub znacznika charakterystyki - wykreślając jedną lub większa liczbę charakterystyk można określić kolor linii lub formę znacznika wykresu. Przy kilku charakterystykach atrybut ten zapisuje się jako listę wartości rozdzielonych spacją. Kolor linii jako wartość dodatnia, a znacznik jako wartość ujemną, jak pokazano w tabeli 4 i przykładzie poniżej.

Tab. 4: Paramery atrybutu wizualizującego wykres plot2d().

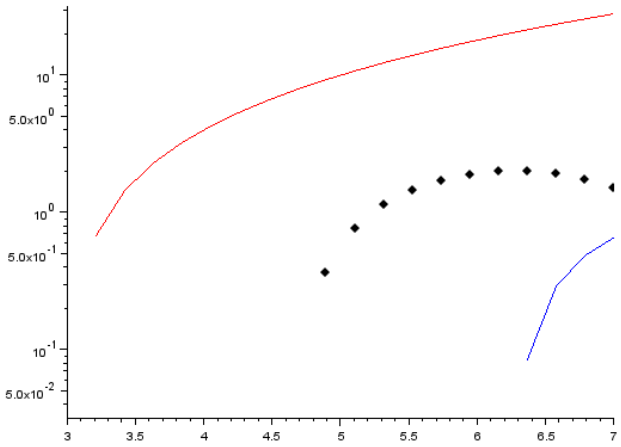
Kolor	Parametr	Znacznik	Parametr
czarny	1	+	-1
niebieski	2	×	-2
zielony	3	⊕	-3
niebieski 2	4	♦	-4
czerwony	5	◇	-5
różowy	6	△	-6
czerwony 2	7	▽	-7
biały	8	⊗	-8
granatowy	9	◯	-9
granatowy 2	10	∗	-10
granatowy 3	11	□	-11

```
clear;clc;close;
x=linspace(3,7,20);
y1=sin(x);
y2=2*cos(x);
y3=0.5*x^2-3*x;
plot2d(x,[y1' y2' y3'],[2 -4 5]
```



- skalowanie osi - osie wykresu mogą być wykreślane w skali normalnej "n"lub logarytmicznej "l". Jest to realizowane poprze przypisanie odpowiedniej wartości do flagi `logflag`, jak pokazano w poniższym przykładzie.

```
clear;clc;close;
x=linspace(3,7,20);
y1=sin(x);
y2=2*cos(x);
y3=x^2-3*x;
plot2d(x,[y1' y2' y3'],
       [2 -4 5], logflag="nl");
```



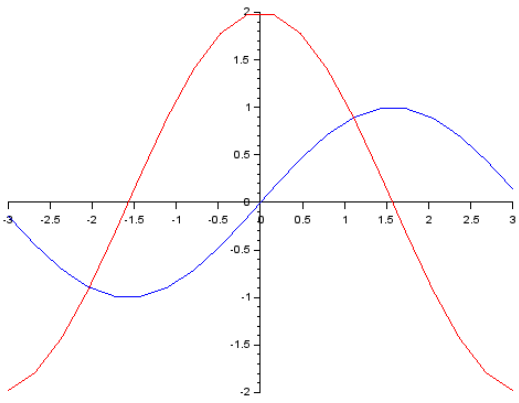
- położenie osi ustawiane jest za pomocą flagi `axesflag` na wartość tożsamą z jedną z podanych w tabeli 5, jak pokazano w przykładowym kodzie zapisanym poniżej.

Tab. 5: Wartości flagi axesflag.

Flaga	Opis
0	Osie nie są rysowane
1	Oś Oy po lewej stronie, wykres otoczony ramką
2	Wykres otoczony ramka bez osi
3	Oś Oy po prawej stronie
4	Osie Ox i Oy przecinają się na środku okna graficznego
5	Osie Ox i Oy przecinają się na środku okna graficznego, wykres otoczony ramką
9	Tryb domyślny, oś Oy po lewej stronie

Niestety za pomocą flagi `axesflag` nie można wykreślić wykresu z osiami Ox i Oy przecinającymi się w punkcie (0,0). Można to jednak osiągnąć odwołując się do uchwytu do osi wykresu za pomocą funkcji `gca()`, jak pokazano poniżej.

```
clear;clc;close;
x=linspace(-3,3,20);
y1=sin(x);
y2=2*cos(x);
plot2d(x,[y1' y2'],[2 5]);
osie=gca();
osie.x_location="origin";
osie.y_location="origin";
```

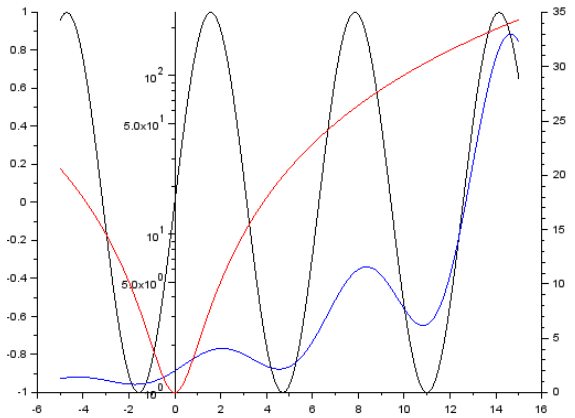


Możliwe do ustawienia położenia osi na wykresie to:

- po lewej - "left"
- po prawej - "right"
- na środku - "middle"
- punkcie 0 osi - "origin"

W podobny sposób można wprowadzić i pogrupować charakterystyki na wykresie przydzielając do wybranych przebiegów niezależne osie. W takim wypadku każdy z zestawów charakterystyk (pod względem przypisania do osi) musi być rysowany osobno, poprzez dodanie nowego uchwytu do osi za pomocą funkcji `newaxes()`. W sytuacji takiej ponieważ wykresy są rysowane jeden na drugim koniecznej odpowiednie pozycjonowanie lub ukrywanie osi i w górnych warstwach wyłączanie tła wykresu. Przykład tak wygenerowanego wykresu pokazano poniżej.

```
clear;clc;clf();
x=linspace(-5,15,200);
y1=sin(x);
y2=exp(x/6).*(y1+2);
y3=1+x.^2;
plot2d(x,y1,1)
os1=gca();
os2=newaxes();
  plot2d(x,y2,2)
  os2.filled="off";
  os2.axes_visible(1)="off";
  os2.y_location="right";
os3=newaxes();
  plot2d(x,y3,5, logflag="nl")
  os3.filled="off";
  os3.axes_visible(1)="off";
  os3.y_location="origin";
```



- Identyfikacja powiązania osi z wykresem w takiej sytuacji może być kłopotliwa, z tego względu należy zawsze uzupełnić wykres o właściwe podpisanie osi i dodanie legendy do wykresu.

  Podpis osi realizuje funkcja `xlabel()`. Pozwala ona na:

  - umieszczenie ciągu tekstowego popisu
  - określenie jego pozycji - flaga `'position'` - na podstawie skalowania osi
  - ustawieni orientacji podpisu, kąta pochylenia - flaga `'rotation'` - w stopniach

– koloru czcionki - flaga `'color'`

– rozmiaru czcionki - flaga `'fontsize'`

– inne dostępne w dokumentacji[1]

Legendę zawierającą opis charakterystyk w wykresu dodaje się za pomocą funkcji `legends()` po wykreśleniu wszystkich charakterystyk. Funkcja ma dwa obowiązkowe atrybuty i jeden opcjonalny:

– wektor nazwa charakterystyk

– wektor styli (kolor linii lub forma znacznika)

– flagę `opt=""` określająca pozycje wykreślenia legendy na wykresie zgodnie z opisem pokazanym w tabeli 6 poniżej.

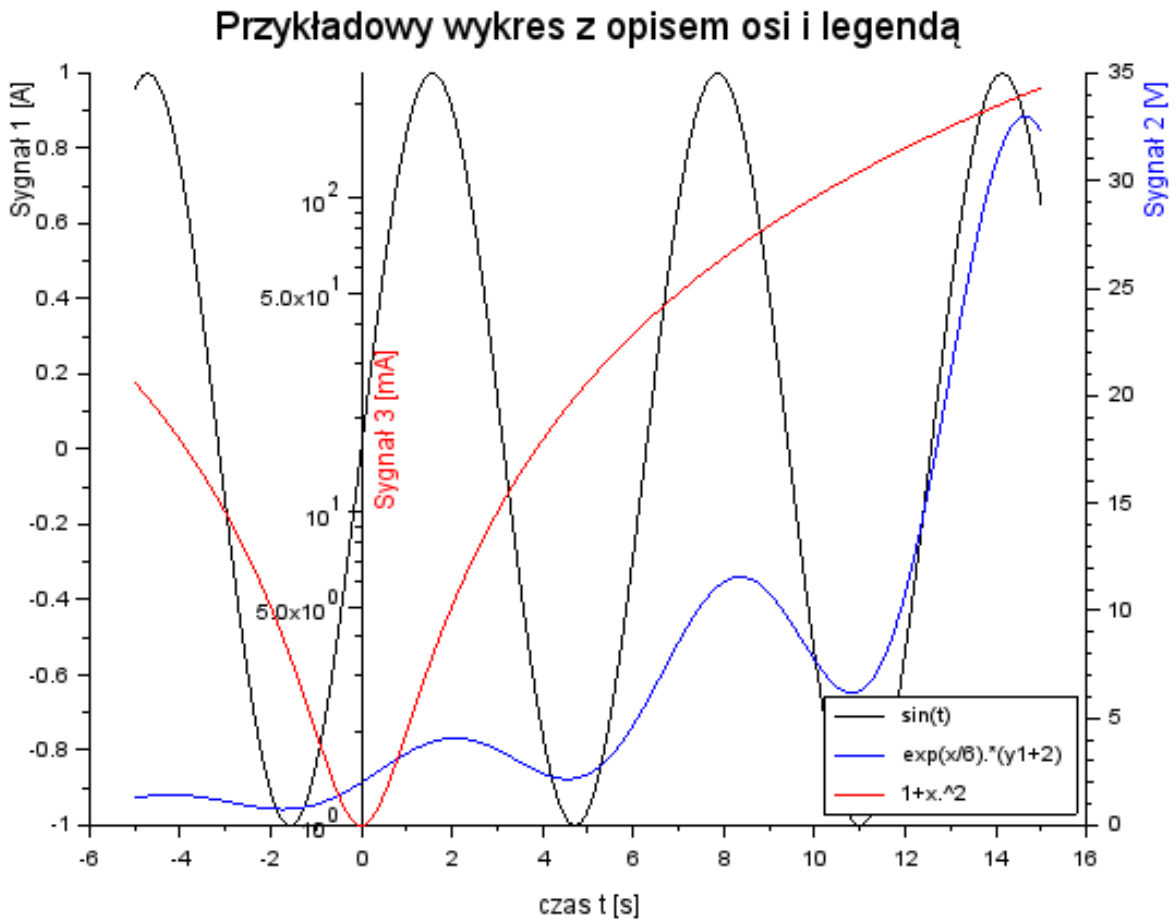
Tab. 6: Wartości flagi pozycjonującej legendę na wykresie.

Opis	Cyfra	Znaki
prawy górny róg	1	"ur"
lewy górny róg	2	"ul"
lewy dolny róg	3	"ll"
prawy dolny róg	4	"lr"
wybór użytkownika (kliknięcie)	5	"?"
pod wykresem	6	"below"

Tytuł wykresu wstawia się za pomocą funkcji `tile()` ustawiając jego treść, formę i pozycję w sposób identyczny jak przy podpisywaniu osi. Przykład implementacji powyższych funkcji przy tworzeniu wykresów pokazano poniżej.

```
clear;clc;clf()
x=linspace(-5,15,200);
y1=sin(x);
y2=exp(x/6).*(y1+2);
y3=1+x.^2;
plot2d(x,y1,1)
xlabel('czas t [s]')
ylabel('Sygnał 1 [A]', 'color',1,'position',[-7,0.6])
os1=gca();
os2=newaxes();
plot2d(x,y2,2)
os2.filled="off";
os2.axes_visible(1)="off";
os2.y_location="right";
ylabel('Sygnał 2 [V]', 'color',2,'position',[18,28])
```

```
os3=newaxes();
plot2d(x,y3,5, logflag="nl")
ylabel('Sygnał 3 [mA]', 'color',5,'position',[1,2])
os3.filled="off";
os3.axes_visible(1)="off";
os3.y_location="origin";
title('Przykładowy wykres z opisem osi i legendą','fontsize',4)
legenda=['sin(t)';'exp(x/6).*(y1+2)';'1+x.^2'];
legends(legenda,[1 2 5], opt="lr");
```



8.2 Wykres powierzchniowy

Jeżeli wykreślana charakterystyka jest funkcją dwóch zmiennych można ją narysować w postaci wykresu *3D* lub powierzchniowego za pomocą funkcji `surf()`. Podstawowe wywołanie funkcji, wymaga podania tylko danych dla wykresu, czyli wektorów opisujących osie *Ox* i *Oy* i macierzy wartości dla osi *Oz*. Zazwyczaj konieczne jest przygotowanie macierzy wartości związanych z wektorami opisującymi osi *Ox* i *Oy*. Jest to realizowane za pomocą funkcji `meshgrid()`, jak pokazano poniżej.

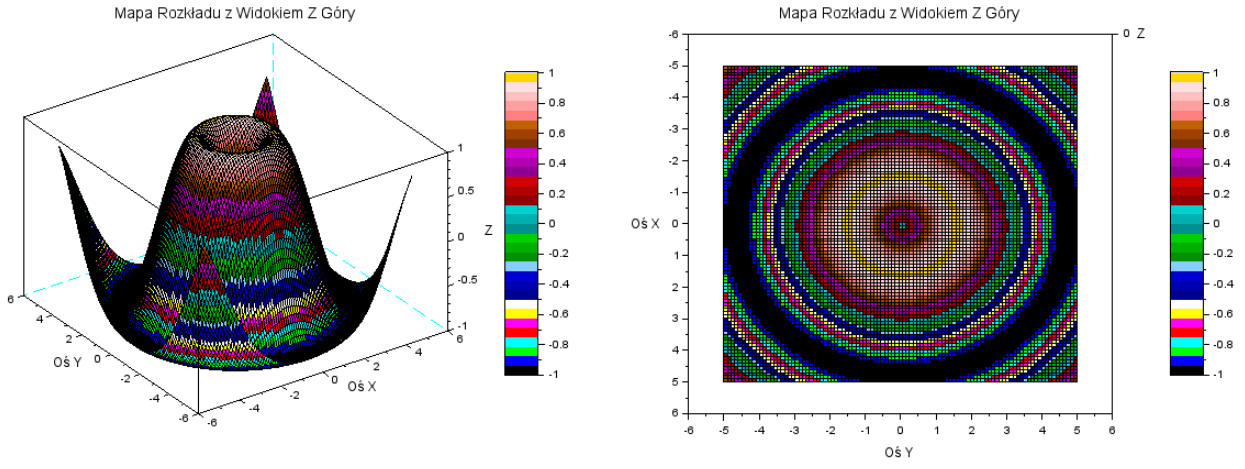
```
[x, y] = meshgrid(x, y);
```

Funkcja `surf()` generuje wykres *3D*, i może on być obracany po uchwyceniu go myszką. Natomiast w wielu sytuacjach oczekiwane może być wykreślenie mapy z widokiem z góry. Realizuje się to za pośrednictwem uchwytu osi wykresu funkcją `gca()` i wywołaniem obrotu wykresu względem osi Ox i Oy, jak pokazano poniżej.

```
gca().rotation_angles = [0 0];
```

Poniżej pokazano przykład generujących wykres *3D* w dwóch przykładowych widokach.

```
clear;clc;close();
x = linspace(-5, 5, 100);
y = linspace(-5, 5, 100);
[x, y] = meshgrid(x, y);
z = sin(sqrt(x.^2 + y.^2));
subplot(1,2,1);
surf(x, y, z);
xlabel('Oś X');
ylabel('Oś Y');
title('Mapa Rozkładu z Widokiem Z Góry');
colorbar();
subplot(1,2,2);
surf(x, y, z);
xlabel('Oś X');
ylabel('Oś Y');
title('Mapa Rozkładu z Widokiem Z Góry');
colorbar();
gca().rotation_angles = [0 0];
gca().y_location="right";
```



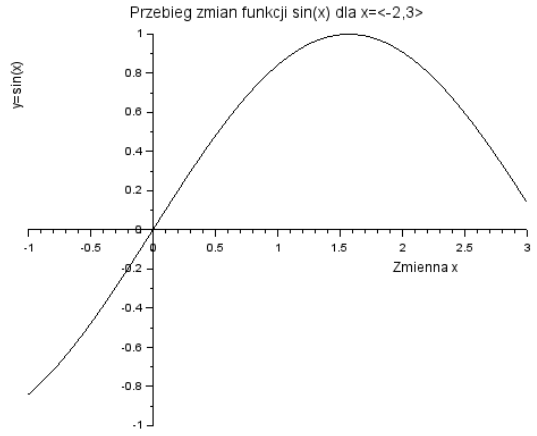
Szczegółowe informacje o składni funkcji można znaleźć w pomocy programu [1].

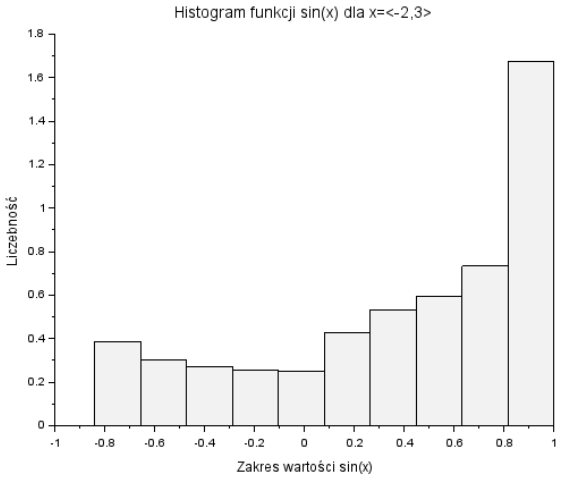
8.3 Histogram

Ostatnim typem omawianego w instrukcji wykresu jest histogram graficznie prezentujący rozkład analizowanej tablicy danych w postaci szeregu umieszczonych na osi prostokątów. Wielkość prostokąta zależy od zdefiniowanego podziału wykresu (liczby słupków, zakresowi wartości w ramach każdego z nich) i wartości w analizowanej tablicy. Wykreślony histogram ułatwia analizę danych zgromadzonych w tablicy (wektorze) informując o częstości występowania wartości z danego przedziału. Przykład zastosowania histogramu pokazano poniżej.

```
clear;clc;close();
x=linspace(-1,3,1000)
dane = sin(x);
subplot(1,2,1);
  plot2d(x,dane);
  gca().y_location="origin";
  gca().x_location="origin";
  xlabel('Zmienna x','position',[1.9,-0.25]);
  ylabel('y=sin(x)','position',[-1,0.6]);
  title('Przebieg zmian funkcji sin(x) dla x=<-2,3>');
subplot(1,2,2);
  histplot(10,dane); //lub histplot(10,dane, normalization=%f);
  xlabel('Zakres wartości sin(x)');
  ylabel('Liczebność');
  title('Histogram funkcji sin(x) dla x=<-2,3>');
```

Dodanie atrybutu z flagą `normalization=%f` spowoduje wyskalowanie osi *Oy* histogramu w liczbie wystąpień w danym zakresie wartości, co wydaje się czytelniejszym rozwiązaniem.





Zadania

Wykorzystując programowanie funkcyjne w środowisku ***Scilab*** napisać skrypty realizujące poniższe zadania:

1. Napisać skrypt obliczający wartość funkcji:

$$f(x) = \begin{cases} e^x & \text{dla } -2 < x \leq 0 \\ \sqrt[3]{x(x+1)} & \text{dla } 0 < x \leq 2 \\ \frac{1}{x} & \text{dla } x \leq -2 \text{ i } x > 2 \end{cases}$$

2. Napisać skrypt obliczający odchylnie sztandarowe zbioru liczb wyznaczonego dla podanego przez użytkownika przedziału na podstawie funkcji z zadania pierwszego. Funkcję odliczającą odchylenie standardowe odnaleźć z wykorzystaniem funkcji `help`.

3. Napisać skrypt wykreślający przebieg funkcji z przykładu pierwszego w zadanym przedziale, oraz histogram dla tego przedziału. Wykreślić oba wykresy w jednym oknie graficznym, podpisać wykresy i osie.

4. Napisać skrypt wyznaczający najbliższe liczby pierwsze do podanej przez użytkownika (mniejsza i większą od podanej liczby)

5. W oparciu o "Równanie artylerzysty" napisać skrypt wykreślający tor lotu pocisku o zadanej prędkości początkowej i kącie wystrzału. program symuluje wystrzały dla trzech katów i dwóch rożnych prędkości początkowych. Serie dla każdej prędkości początkowej przypisane są do dwóch osi *Oy* (jedna po lewej a druga po prawej tronie wykresu). Jedna seria wkreślana jest za pomocą znaczników a druga za pomocą krzywej o zadanym kolorze. Opisać osie i wykres, dodać legendę.

Bibliografia

[1] *Strona Scilab Enterprises S.A.S.* 2023. URL: `http://www.scilab.org`.

[2] *Wbudowany podręcznik programowania (help) SciLab.*