Podstawy programowania w Scilabie

Wstęp

Korzystanie ze środowiska obliczeniowego ***Scilab*** wyłącznie za pośrednictwem konsoli jest mało efektywne. Wygodniejszym rozwiązaniem jest zapisanie sekwencji instrukcji w pliku tekstowym, a następnie przekazanie tego pliku do wykonania. Środowisko udostępnia narzędzia wspierające tworzenie programów. Zapisywanie sekwencji obliczeniowych w skryptach upraszcza opracowywanie i późniejsze wykorzystywanie algorytmu. Kod można pisać w dowolnym edytorze tekstowym, jednak efektywniejsza jest praca w edytorze *SciNotes*, zintegrowanym ze środowiskiem. Skrypt wykonywalny powinien mieć rozszerzenie **.sce*; wbudowany edytor nadaje je automatycznie.

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.

Algorytm zapisany na dysku w postaci skryptu może być wykonywany wielokrotnie. Podobnie jak w językach ogólnego przeznaczenia, można tworzyć pliki zawierające zestawy funkcji, które następnie dołącza się do skryptów obliczeniowych. Takie pliki zapisuje się z rozszerzeniem **.sci*. Skrypt można wykonać w konsoli za pomocą instrukcji `exec()`; ta sama instrukcja służy do wczytania funkcji zdefiniowanych w pliku **.sci*.

1 Skrypt

Skrypt w środowisku ***Scilab*** nie wymaga obowiązkowego szablonu, ponieważ jest ciągiem instrukcji wykonywanych kolejno przez interpreter. Ze względów praktycznych warto jednak stosować stały układ złożony 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 pokazano poniżej. Należy zwrócić uwagę na dwa sposoby dodawania komentarzy: jedno- i wielowierszowych. Komentarze ułatwiają innym użytkownikom analizę kodu, a autorowi – jego późniejszą modyfikację. Pozwalają także tymczasowo wyłączyć wybrane fragmenty kodu z wykonania przez interpreter.

Tab. 1. Tryby działania instrukcji exec()

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

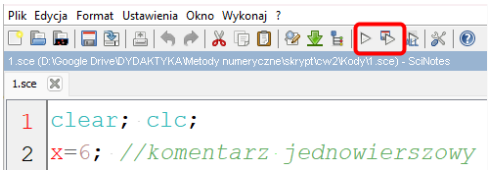
0	wyświetlany jest efekt działania linii nie zakończonych średnikiem	`z =`
		`4.`
		`-->`

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	

```
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
```

Powyższy skrypt można wykonać w edytorze *SciNotes* za pomocą przycisku *Wykonaj* lub *Zapisz i wykonaj*, jak pokazano na ryc. 1, albo za pomocą polecenia `exec()` w konsoli. Sposób prezentacji poleceń i wyników zależy od trybu wykonania oraz od średników kończących wiersze. Drugim argumentem funkcji `exec()` można sterować poziomem komunikatów; jego wartości i efekty zestawiono w Tab. 1.



Ryc. 1. Przyciski wykonania skryptu z poziomu edytora SciNotes

Przed wykonaniem skryptu z konsoli należy upewnić się, że aktywny katalog roboczy zawiera odpowiedni plik. Bieżący katalog można sprawdzić poleceniem `pwd`, a zmienić funkcją `cd`. Można go także wygodnie przełączyć za pośrednictwem przeglądarki plików znajdującej się po lewej stronie konsoli.

2 Instrukcje interfejsu użytkownika

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=");
```

- wyświetlanie wyników obliczeń
  W zależności od specyfiki zadania realizowanego przez skrypt wyniki mogą być zwracane na trzy sposoby:
  - dane liczbowe w konsoli,
  - graficznie w postaci wykresu,
  - zapis danych do pliku.

  Pierwszy sposób można realizować za pomocą funkcji `disp()` i `mprintf()`, omówionych w rozdziale 1, jak pokazano w poniższym skrypcie. Prezentację graficzną opisano w dalszej części niniejszego rozdziału, a zapis do pliku – w sekcji dotyczącej operacji wejścia i wyjścia.

```
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 pierwszym wywołaniu funkcji `disp()` zbudowano jeden ciąg tekstowy, natomiast w drugim każdy argument rozdzielony przecinkiem jest obliczany i wyświetlany niezależnie.

3 Decyzje

W kodzie ***Scilab*** można sterować wykonaniem instrukcji za pomocą warunków i instrukcji wyboru. Decyzja jest podejmowana na podstawie wartości wyrażenia logicznego, budowanego z operatorów relacji i operatorów logicznych. Zestawienie tych operatorów przedstawiono w Tab. 2.

Operatory pojedyncze `&` i `|` działają element po elemencie na tablicach logicznych. Operatory podwójne `&&` i `||` zwracają pojedynczą wartość logiczną i stosują ocenę skróconą: prawy argument jest obliczany tylko wtedy, gdy jest potrzebny do ustalenia wyniku.

Podstawową konstrukcją sterującą jest instrukcja warunkowa o schemacie `if warunek then kod end`. Kiedy wyrażenie zapisane po słowie `if` zwróci wartość prawdziwą (`%t`), wykonany zostaje kod umieszczony między słowami kluczowymi `then` i `end`.

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.

Tab. 2. Operatory relacji i logiczne

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

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

Jeżeli zostanie podana liczba dodatnia, program wygeneruje macierz losową, jak pokazano poniżej; w przeciwnym razie 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 w innych językach programowania, można zaprogramować reakcję na niespełnienie warunku logicznego. W tym celu konstrukcję uzupełnia się słowem kluczowym `else`, po którym umieszcza się kod wykonywany dla wartości fałszywej, jak w poniższym przykładzie.

```
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
```

Instrukcję warunkową `if` można rozbudować o kolejne gałęzie za pomocą konstrukcji `elseif warunek then`. Kaskada może zawierać kilka gałęzi realizujących różne zadania. Należy pamiętać, że bez końcowej gałęzi `else` może nie zostać wykonany żaden blok; z gałęzią `else` zostanie wykonany dokładnie jeden z nich. Poniżej pokazano przykładowy skrypt z kaskadą warunkową.

```
clear; clc;
```

```
x=input("Określ rozmiar macierzy=");
if x>0 & x<6 then
  Z=rand(x,x);
  disp("Macierz Z:",Z);
elseif x>=6 then
  disp("Podano za duży rozmiar macierzy,");
  disp("wygenerowana zostanie macierz w dopuszczalnym
        rozmiarze n=5.");
  Z=rand(5,5);
  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=5."
  "Macierz Z:"
   0.4204123   0.251896    0.0417362   0.0540932   0.8387927
   0.4277572   0.4391129   0.3438272   0.9190207   0.4343749
   0.3184586   0.0759304   0.1970167   0.4603516   0.7767876
   0.5761894   0.255938    0.2122899   0.2992685   0.1395318
   0.4254902   0.0670617   0.3140399   0.0029166   0.1150637
```

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

Rozbudowana kaskada `if elseif` może być nieczytelna. Jeżeli kryterium decyzyjnym jest wartość jednej zmiennej, korzystniej zastosować instrukcję wyboru. W środowisku ***Scilab*** służy do tego konstrukcja `select case`, złożona z trzech części:

- słowo `select` wskazujące zmienną 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");
```

```
select 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
```

W powyższym kodzie użyto zarówno apostrofów, jak i cudzysłowów. Scilab dopuszcza oba delimitery ciągów tekstowych; dla czytelności należy wybrać jedną konwencję i stosować ją konsekwentnie. Drugi rodzaj delimitera jest przydatny, gdy ma wystąpić wewnątrz tekstu.

5 Pętle

Możliwość wielokrotnego wykonywania kodu jest kolejną cechą języków programowania dostępną w środowisku ***Scilab***. Stosuje się dwie podstawowe konstrukcje pętli: iteracyjną i warunkową.

5.1 Pętla iteracyjna `for`

Pętla iteracyjna wykonuje powiązany blok kodu dla kolejnych elementów zadanego wektora. Liczba powtórzeń jest zatem równa liczbie wartości przypisywanych zmiennej iteracyjnej. Rozwiązanie to przypomina pętlę `for` w języku Python oraz pętlę zakresową 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 łączy powtarzanie bloku kodu z kontrolą warunku logicznego. Wykonuje blok tak długo, jak długo warunek pozostaje spełniony. W środowisku ***Scilab*** dostępne są trzy równoważne formy deklaracji pętli warunkowej, pokazane poniżej.

```
while warunek then        while warunek do          while warunek,
   kod;                      kod;                      kod;
end                       end                       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 można sterować przebiegiem pętli za pomocą instrukcji `continue` i `break`. Pierwsza przerywa aktualną iterację i przechodzi do sprawdzenia warunku lub pobrania kolejnej wartości sterującej. Druga kończy całą pętlę i przekazuje sterowanie do pierwszej instrukcji zapisanej za nią. Instrukcje te stosuje się zwykle wewnątrz instrukcji warunkowej `if`.

6 Funkcje

Kod skryptu w ***Scilab***, podobnie jak w innych językach, można podzielić pod kątem realizowanych zadań, wydzielając powtarzalne fragmenty do funkcji. Poprawia to czytelność i ułatwia ponowne wykorzystanie kodu. Funkcje używane w wielu skryptach warto umieścić w osobnym pliku `*.sci` i wczytywać go na początku programu.

Funkcje lokalne można deklarować na początku kodu, po instrukcjach inicjujących, ale przed właściwą częścią skryptu. W ***Scilab*** funkcję można zdefiniować na dwa sposoby:

1. pełny – w bloku `function ... endfunction`. Definicja określa listę wartości zwracanych, nazwę funkcji i listę argumentów, a po niej następuje kod wykonywany przez funkcję. Funkcje w ***Scilab*** mogą zwracać zarówno pojedyncze wartości, jak i tablice. Poniżej pokazano przykład deklaracji i wywołania 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*** może zwracać więcej niż jedną wartość. Wartości te mogą mieć różne typy i mogą być tablicami. Przy wywołaniu funkcji listę wyników zapisuje się w nawiasach kwadratowych i przypisuje odpowiednim zmiennym. Przykład funkcji zwracającej 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 ułatwia tematyczne grupowanie funkcji w plikach bibliotecznych `*.sci` i wczytywanie ich na początku skryptu, jak pokazano w poniższym przykładzie. Należy pamiętać, że trzeba podać poprawną ścieżkę do pliku albo ustawić katalog roboczy ***Scilab*** na folder zawierający bibliotekę.

```
//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:length(L)
      disp("sin("+string(L(i))+")="+string(S(i)));
    end
  endfunction
// plik skryptu skrypt.sce
  clear; clc;
```

```
exec("lib/fun.sci",-1);
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 – za pomocą funkcji `deff()`. Efekt jest równoważny użyciu konstrukcji `function`, ale zapis jest przeznaczony dla prostych funkcji o krótkiej definicji. Funkcja `deff` przyjmuje dwa argumenty rozdzielone przecinkiem:

   - deklaracja wywołania funkcji, analogiczna do konstrukcji `function`: `[wyniki]=nazwa(argumenty)`,
   - kod funkcji zapisany jako ciąg tekstowy.

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

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

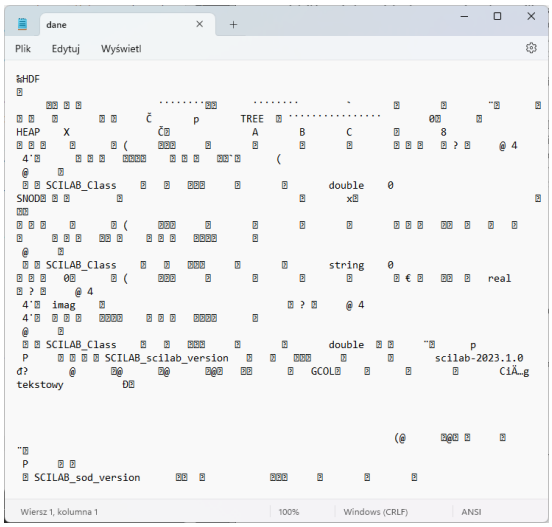
7 Zapis i odczyt danych z pliku

Dane przechowywane w zmiennych znajdują się w pamięci operacyjnej, a więc są ulotne. Aby zachować wyniki, można je zapisywać do plików i później odczytywać. ***Scilab*** obsługuje zapis oraz odczyt danych w trybie binarnym, a także zapis do pliku tekstowego w postaci czytelnej dla człowieka.

Zapis w trybie binarnym realizuje funkcja `save()`. Pierwszy argument wskazuje plik na dysku, a kolejne tworzą listę zmiennych przeznaczonych do zapisania. Należy pamiętać, że ścieżka względna jest interpretowana względem aktualnego katalogu roboczego ***Scilab***. Trzeba więc podać poprawną ścieżkę względną lub bezwzględną albo wcześniej ustawić właściwy katalog roboczy.

Odczyt w trybie binarnym realizuje funkcja `load()`. Ma ona składnię podobną do funkcji zapisu, jednak lista zmiennych jest opcjonalna. Funkcja `save()` zapisuje zarówno dane, jak i nazwy przekazanych do niej zmiennych. Po pominięciu listy funkcja `load()` odtworzy wszystkie zapisane zmienne wraz z nazwami. Możliwy jest również odczyt selektywny, ale można wskazać wyłącznie nazwę istniejącą w pliku. Poniżej zamieszczono przykład zapisu i odczytu danych w trybie binarnym, a na ryc. 2 pokazano podgląd utworzonego pliku.

```
clear;clc;
A=[1,2,3,4,5];
B="Ciąg tekstowy";
C=12+3*%i;
```



Ryc. 2. Plik binarny z danymi z przykładu

```
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 realizuje funkcja `print()`. Operacja przypomina wyświetlanie danych w konsoli za pomocą funkcji `disp("nazwa", lista_zmiennych)`. Dla pliku tekstowego podaje się nazwę pliku; rozszerzenie jest nadawane automatycznie i zapisywana jest wyłącznie zawartość zmiennych.

Wiele urządzeń pomiarowych umożliwia eksport danych do pliku w formacie `*.csv`. ***Scilab*** importuje takie dane za pomocą funkcji `csvRead()`. Poprawne skonfigurowanie argumentów wymaga sprawdzenia struktury pliku i położenia danych, które mają zostać zaimportowane. Należy także ustalić separator dziesiętny: w zapisie domyślnym Scilab używa kropki, dlatego dane z przecinkiem dziesiętnym mogą wymagać odpowiedniego ustawienia argumentu lub konwersji. W podstawowej postaci funkcja wymaga nazwy pliku i separatora kolumn, jak pokazano poniżej.

```
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
  0.09 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 pominiętym argumentom funkcji. Ostatni argument określa zakres importowanych danych:

- 2 – numer wiersza, od którego rozpoczyna się import danych,
- 1 – numer kolumny, od której rozpoczyna się import danych,
- 11 – numer wiersza, 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*** można zapisać w pliku `*.csv` za pomocą funkcji `csvWrite()`, która przyjmuje tablicę danych i nazwę pliku. Tablicę można przekazać jako pojedynczą zmienną lub zestaw wektorów, jak pokazano poniżej. Trzeci argument określa separator kolumn, a czwarty – separator dziesiętny. Ich dobór zależy od programu, w którym plik będzie później otwierany. Dla polskich ustawień programu Excel można użyć średnika jako separatora kolumn i przecinka jako separatora dziesiętnego. Po pominięciu obu argumentów Scilab stosuje domyślnie przecinek między kolumnami i kropkę dziesiętną.

```
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 omawianym, a zarazem jednym z najważniejszych sposobów prezentacji wyników jest przedstawienie danych na wykresach. ***Scilab*** udostępnia wiele typów wykresów; w zastosowaniach inżynierskich szczególnie przydatne są:

- wykres punktowy (XY) z interpolacją – przebieg zmian analizowanej wielkości,
- wykres powierzchniowy ($3D$) – mapa rozkładu analizowanej wielkości,
- histogram – analiza rozkładu empirycznego badanej wielkości.

Każdy wykres powinien, oprócz właściwie dobranej formy prezentacji danych, zawierać:

- czytelny tytuł informujący o prezentowanych danych,
- odpowiednio wyskalowane i opisane osie,
- jednoznaczne oznaczenia przebiegów oraz legendę, jeżeli przedstawiono więcej niż jedną charakterystykę.

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

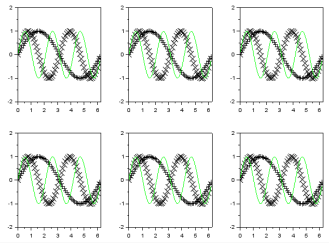
- clf; - czyszczenie zawartości otwartych okien graficznych
- close(); - kasowanie otwartych okien graficznych

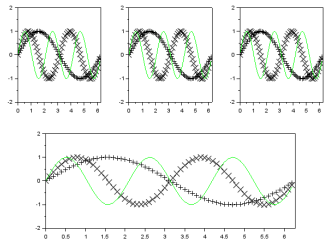
W wielu sytuacjach dane z symulacji trzeba zaprezentować na kilku wykresach. ***Scilab*** pozwala rozmieścić określoną liczbę wykresów w jednym oknie graficznym. Drugą opcją jest otwarcie osobnego okna dla każdego wykresu.

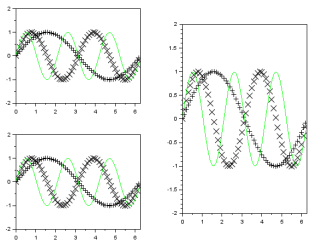
Podział pojedynczego okna na kilka wykresów realizuje funkcja `subplot()`. Dwa pierwsze argumenty określają liczbę wierszy i kolumn siatki, a trzeci aktywuje wybrane pole. Numeracja zaczyna się od lewego górnego pola i przebiega wierszami. W Tab. 3 zestawiono przykładowe układy. Kolejne wywołania mogą nakładać na siebie siatki o różnych podziałach.

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 wcześniej utworzonego okna – `scf(nazwa);` lub `scf(ID);`.
- Skasowanie wybranego okna graficznego –> `close(nazwa);` lub `close(ID);`

Poniższy przykład ilustruje omówione instrukcje. Skrypt tworzy dwa okna graficzne i kolejno umieszcza w nich wykresy. Następnie oba okna są zamykane, po czym ponownie otwierane jest okno o *ID = 2* i rysowany w nim wykres.

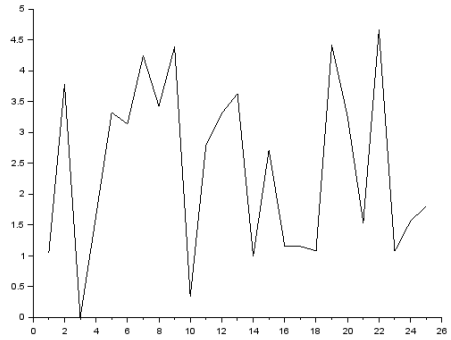
```
clear;clc;close();
okno_1=scf(1);
okno_2=scf(2);
  plot2d();
scf(okno_1);
  subplot(1,2,1); plot2d();
  subplot(1,2,2); plot2d();
close(okno_2); close(1);
scf(2); plot2d();
```

8.1 Wykres punkowy

Jest to podstawowa forma prezentacji danych pomiarowych i symulacyjnych. Pozwala wykreślić zestaw punktów w przestrzeni *2D*. Wykres generuje funkcja `plot2d()`, opisy osi i tytuł – `xtitle()`, a legendę – `legend()` lub `legends()`.

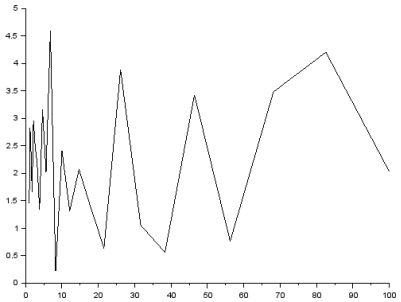
Podstawowa forma wywołania funkcji `plot2d()` wymaga przekazania jako atrybutu wektora z danymi do wykreślenia. W takim przypadku dane względem osi *Ox* zostaną wykreślone względem pozycji tablicy, jak pokazano poniżej.

```
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);
```

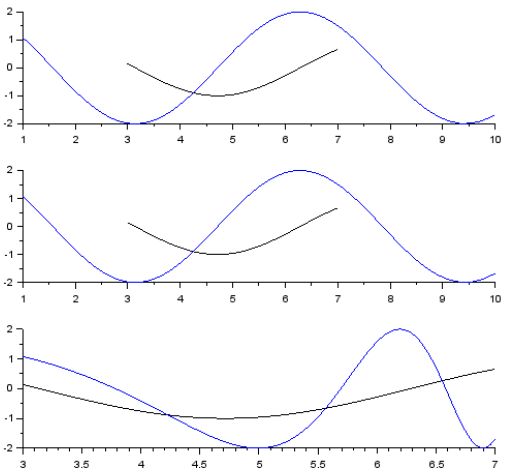


Funkcja `plot2d()` umożliwia umieszczanie na wspólnym wykresie kilku charakterystyk. Można to zrealizować na trzy sposoby:

- niezależne wywołanie funkcji `plot2d()` dla każdej charakterystyki; skalowanie osi jest dobierane automatycznie do zakresu 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*.

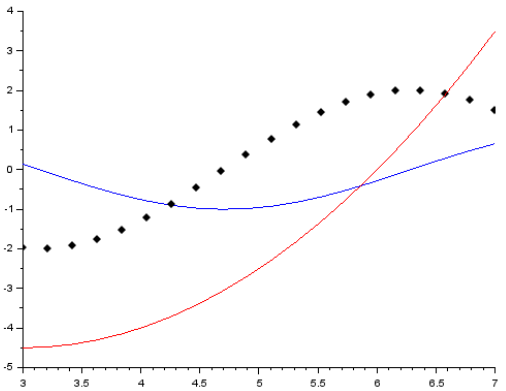
Funkcja `plot2d` posiada jeszcze kilka opcjonalnych atrybutów pozwalających na:

- ustawienie koloru lub znacznika charakterystyki – dla jednej lub większej liczby charakterystyk można określić kolor linii albo formę znacznika. Przy kilku przebiegach argument zapisuje się jako listę wartości rozdzielonych spacjami: wartość dodatnia wybiera kolor linii, a ujemna – znacznik, jak pokazano w Tab. 4 i poniższym przykładzie.

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

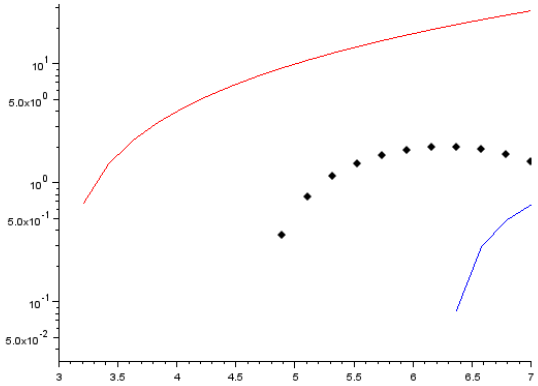
Kolor	Parametr	Znacznik	Parametr
czarny	1	+	-1
niebieski	2	×	-2
zielony	3	⊕	-3
niebieski 2	4	♦	-4
czerwony	5	◇	-5
różowy	6	△	-6
czerwony 2	7	▽	-7
biały	8	⊗	-8
granatowy	9	◯	-9
granatowy 2	10	∗	-10
granatowy 3	11	□	-11

```
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 przedstawione w skali liniowej (`n`) lub logarytmicznej (`l`). Ustawienie realizuje się przez przypisanie odpowiedniej wartości do argumentu `logflag`, jak pokazano poniżej.

```
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");
```



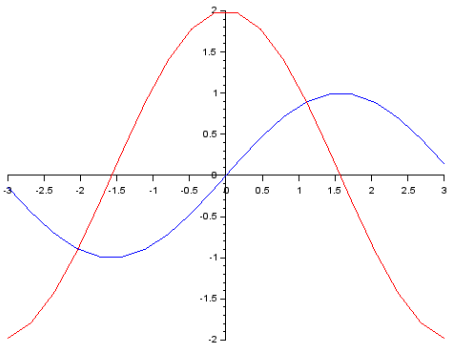
- położenie osi ustawiane jest za pomocą flagi `axesflag` na wartość tożsamą z jedną z podanych w tabeli 5, jak pokazano w przykładowym kodzie zapisanym poniżej.

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";
```

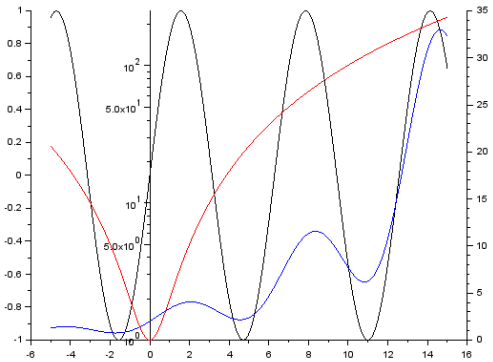


Możliwe do ustawienia położenia osi na wykresie to:

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

W podobny sposób można pogrupować charakterystyki, przydzielając wybranym przebiegom niezależne osie. Każdy zestaw przypisany do innej osi należy narysować osobno, po utworzeniu nowego obiektu osi funkcją `newaxes()`. Ponieważ kolejne obiekty są nakładane na siebie, trzeba ustawić ich położenie, ukryć zbędne elementy osi i wyłączyć tło w warstwach położonych wyżej. Przykład takiego 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
  - ustawienie orientacji podpisu, czyli kąta pochylenia w stopniach – właściwość `'rotation'`.
  - koloru czcionki - flaga `'color'`
  - rozmiaru czcionki - flaga `'fontsize'`
  - inne dostępne w dokumentacji[1]

  Legendę zawierającą opis charakterystyk z wykresu dodaje się za pomocą funkcji `legends()` po narysowaniu wszystkich przebiegów. Funkcja ma dwa argumenty obowiązkowe i jeden opcjonalny:

  - wektor nazwa charakterystyk
  - wektor styli (kolor linii lub forma znacznika)

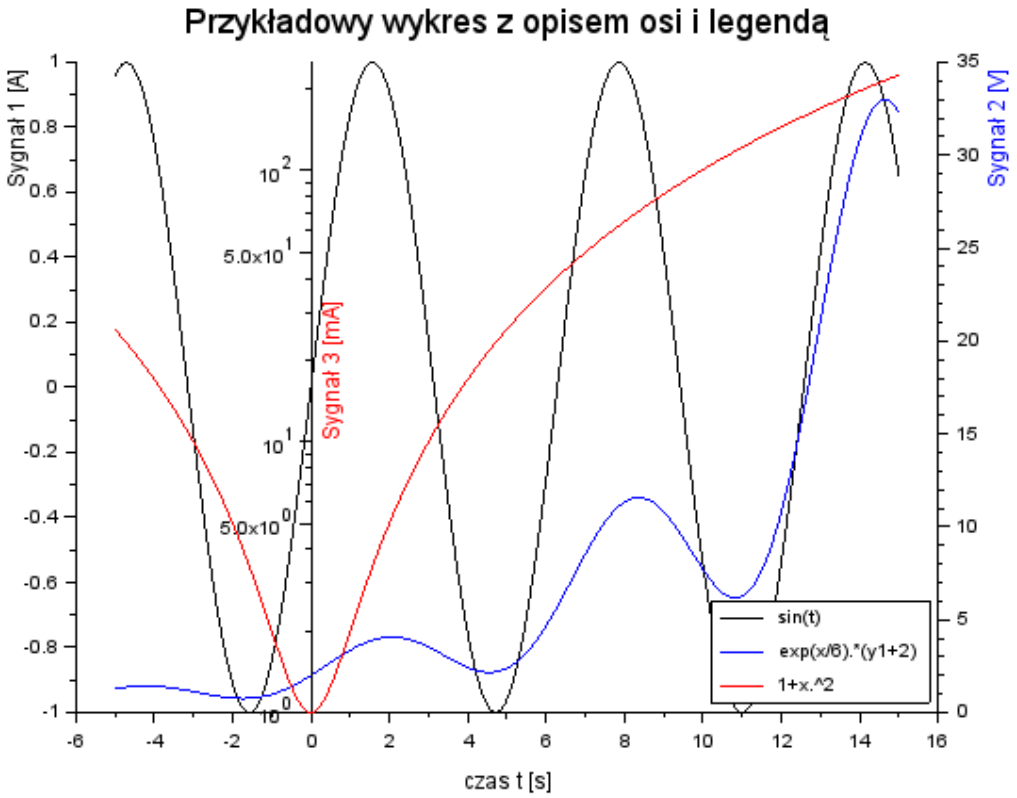
– flagę `opt=""` określającą pozycję legendy zgodnie z opisem w Tab. 6.

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 `title()`, ustawiając jego treść, formę i pozycję podobnie jak w przypadku podpisów osi. Przykład użycia omówionych funkcji 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 charakterystyka jest funkcją dwóch zmiennych, można ją przedstawić jako wykres powierzchniowy *3D* za pomocą funkcji `surf()`. Podstawowe wywołanie wymaga wektorów opisujących osie *Ox* i *Oy* oraz macierzy wartości odpowiadających osi *Oz*. Siatkę współrzędnych związaną z wektorami osi *Ox* i *Oy* można przygotować funkcją `meshgrid()`, jak pokazano poniżej.

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

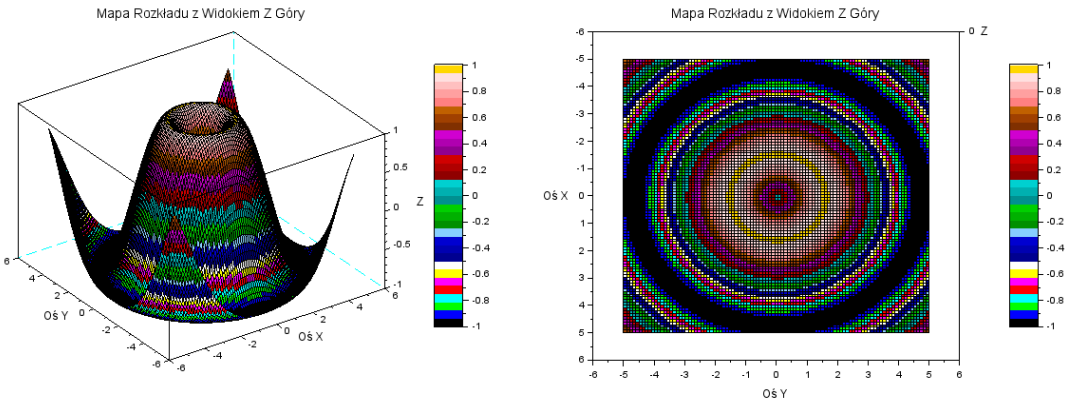
Funkcja `surf()` generuje wykres *3D*, i może on być obracany po uchwyceniu go myszką. Natomiast w wielu sytuacjach oczekiwane może być wykreślenie mapy z widokiem z góry. Realizuje się to za pośrednictwem uchwytu osi wykresu funkcją `gca()` i wywołaniem obrotu wykresu względem osi *Ox* i *Oy*, jak pokazano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

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

Poniżej pokazano przykład generowania wykresu *3D* w dwóch 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";
```



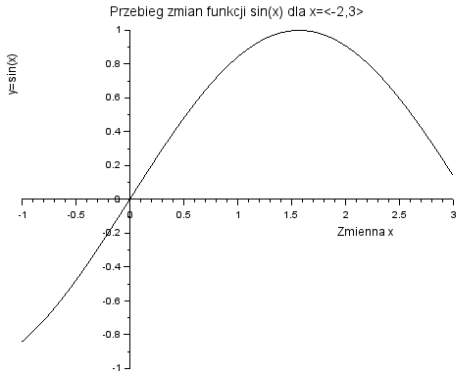
Szczegółowe informacje o składni funkcji można znaleźć w pomocy programu [1].

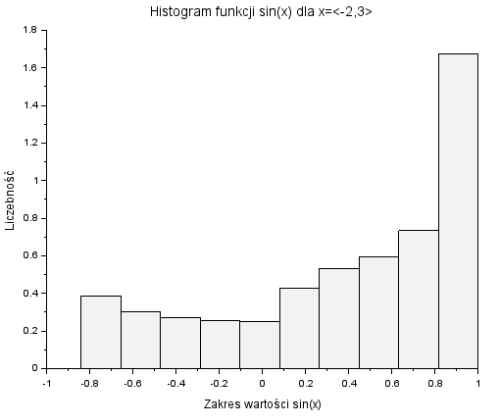
8.3 Histogram

Ostatnim omawianym typem wykresu jest histogram, który przedstawia rozkład danych za pomocą prostokątnych słupków. Ich wysokość zależy od liczby obserwacji w danym przedziale, a szerokość – od przyjętego podziału zakresu wartości. Histogram ułatwia ocenę częstości występowania danych w poszczególnych przedziałach. Przykład 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>');
```

Ustawienie argumentu `normalization=%f` skaluje oś *Oy* histogramu liczbą obserwacji w danym przedziale, co bywa bardziej czytelnym rozwiązaniem.





Zadania

Celem ćwiczenia jest przygotowanie programu generującego i analizującego przebiegi elektryczne. Poszczególne operacje należy zapisać w postaci funkcji, a główny skrypt powinien odpowiadać za wybór zestawu, wywołanie funkcji i prezentację wyników. Każdy student realizuje zestaw o numerze wskazanym przez prowadzącego.

Zestaw	typ	U_m [V]	f [Hz]	φ [°]	T [s]	f_s [Hz]	limit [V]
1	sinus	325	50	0	0,10	5000	–
2	prostokątny	24	100	0	0,05	10000	–
3	ograniczony	10	200	20	0,03	20000	6
4	sinus	170	60	15	0,10	6000	–
5	prostokątny	12	250	30	0,04	12500	–
6	ograniczony	18	120	-20	0,05	12000	11
7	sinus	230	40	45	0,15	4000	–
8	prostokątny	48	75	-30	0,08	7500	–
9	ograniczony	8	400	10	0,025	24000	5
10	sinus	100	125	-45	0,04	10000	–
11	prostokątny	15	300	20	0,03	18000	–
12	ograniczony	30	50	0	0,12	5000	18
13	sinus	50	500	60	0,02	25000	–
14	prostokątny	5	1000	0	0,01	50000	–
15	ograniczony	12	333	-15	0,03	19980	7

1. Pobrać parametry z tabeli dla swojego zestawu. Napisać funkcję generującą wektor czasu i wartości napięcia dla wskazanego przebiegu: sinusoidalnego, prostokątnego albo sinusoidalnego ograniczonego na zadanym poziomie. Argumentami funkcji powinny być co najmniej: amplituda, częstotliwość, faza, czas obserwacji i częstotliwość próbkowania.

2. Za pomocą instrukcji wyboru przygotować interfejs pozwalający wybrać rodzaj przebiegu. Sprawdzić poprawność argumentów i wyświetlić czytelny komunikat w przypadku wartości niedopuszczalnych.

3. Obliczyć wartość średnią, skuteczną i szczytową oraz współczynnik szczytu. Dla sinusoidy porównać wartość skuteczną z wartością analityczną $U_{\mathrm{sk}} = U_m/\sqrt{2}$.

4. W jednym oknie graficznym przedstawić przebieg czasowy i histogram wartości. Opisać osie z jednostkami, dodać tytuły i legendę, a dane zapisać do pliku CSV.

Zadanie rozszerzające. W pętli zbadać wpływ częstotliwości próbkowania na względny błąd numerycznie wyznaczonej wartości skutecznej.

Bibliografia

[1] *Strona Scilab Enterprises S.A.S.* 2024. URL: `http://www.scilab.org`.

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