POLITECHNIKA LUBELSKA

PODRĘCZNIKI

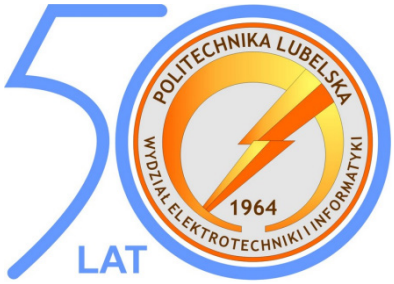
Leszek Jaroszyński
Michał Łanczont

Laboratorium metod numerycznych

Lublin 2014

Laboratorium metod numerycznych

Podręczniki – Politechnika Lubelska



Człowiek – najlepsza inwestycja

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Publikacja współfinansowana ze środków Unii Europejskiej w ramach
Europejskiego Funduszu Społecznego

Leszek Jaroszyński
Michał Łanczont

Laboratorium metod numerycznych

Politechnika Lubelska
Lublin 2014

Recenzenci:
dr inż. Elżbieta Ratajewicz
dr inż. Dariusz Czerwiński

Skład: Michał Łanczont

Podręcznik przeznaczony dla studentów studiów stacjonarnych i niestacjonarnych pierwszego stopnia na kierunku Elektrotechnika na Wydziale Elektrotechniki i Informatyki Politechniki Lubelskiej

Publikacja dystrybuowana bezpłatnie.

Publikacja przygotowana i wydana w ramach projektu „Kwalifikacje dla rynku pracy - Politechnika Lubelska przyjazna dla pracodawcy" nr umowy POKL.04.03.00-00-035/12 z dnia 27 marca 2013 r. współfinansowanego ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego.

Publikacja wydana za zgodą Rektora Politechniki Lubelskiej

ISBN: 978-83-7947-019-8

Wydawca: Politechnika Lubelska
ul. Nadbystrzycka 38D, 20-618 Lublin
Realizacja: Biblioteka Politechniki Lubelskiej
Ośrodek ds. Wydawnictw i Biblioteki Cyfrowej
ul. Nadbystrzycka 36A, 20-618 Lublin
tel. (81) 538-46-59, email: wydawca@pollub.pl
www.biblioteka.pollub.pl
Druk: TOP Agencja Reklamowa Agnieszka Łuczak
www.agencjatop.pl

Elektroniczna wersja książki dostępna w Bibliotece Cyfrowej PL www.bc.pollub.pl
Nakład: 100 egz.

Spis Treści

1. Wprowadzenie do środowiska Scilab

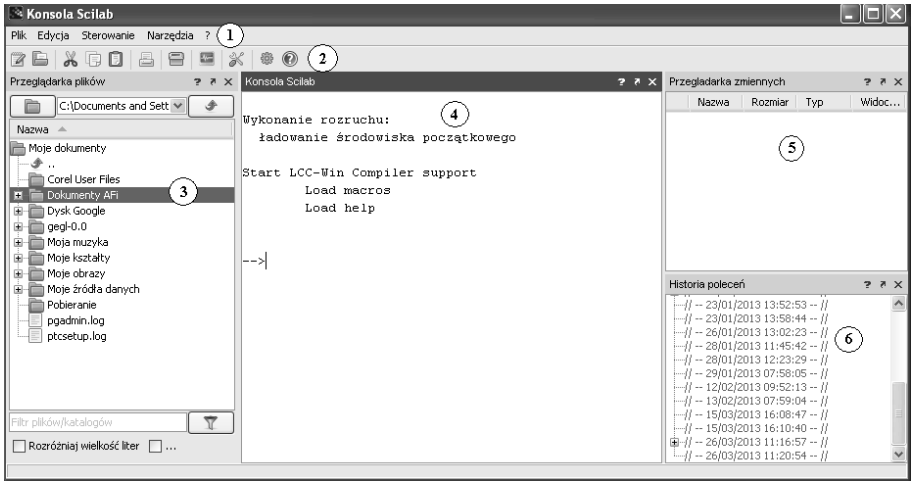
Metody numeryczne są dziedziną matematyki zajmującą się rozwiązywaniem problemów matematycznych na liczbach. Wyniki otrzymywane w efekcie działania określonych procedur są zazwyczaj przybliżone, jednakże możliwe jest określenie z góry wymaganego przybliżenia.

Metody numeryczne znalazły szerokie zastosowanie przy analizie i rozwiązywaniu problemów inżynierskich, w sytuacjach gdy przedmiot badań nie ma rozwiązania analitycznego, określonego wzorem, lub jest ono skomplikowane i czasochłonne.

Programowanie i przeprowadzanie symulacji badanego obiektu lub zjawiska najwygodniej przeprowadzać w wyspecjalizowanym programie matematycznym tj. Matlab, Mathcad, Maple, Maxima i Scilab [1],[3].

Niniejsza książka oraz cykl zajęć prowadzonych na Wydziale Elektrotechniki i Informatyki na studiach stacjonarnych i niestacjonarnych przeprowadzany jest z wykorzystaniem środowiska obliczeniowego Scilab w wersji 5.4 (32 bit). Jest to program podobny do Matlab'a, zawiera opcję importu projektów przygotowanych w środowisku Matlab. Projekt rozwija się dynamicznie, zawiera pokaźną bibliotekę zewnętrznych metod i procedur - stale rozwijaną, oraz jest narzędziem darmowym. Program można pobrać ze strony projektu [2].

Interfejs programu składa się z sześciu części, jak pokazano na rysunku 1.1.



Rys. 1.1. Interfejs programu Scilab (1 – menu główne, 2 – pasek skrótów, 3, 5 i 6 – okna pomocnicze, 4 – okno konsoli)
Źródło: opracowanie własne

Menu główne aplikacji (**1**), podzielono na sześć podmenu: *Plik*, *Edycja*, *Sterowanie*, *Narzędzia* i *Pomoc* oznaczona znakiem „?".

Menu *Plik* zawiera skróty do operacji systemowych takich jak zapis i odczyt sesji, wykonanie zapisanej sesji, ustawienie katalogu roboczego oraz wykonanie wydruku. Wszystkie wymienione powyżej operacje odnoszą się do prac w konsoli (**4**).

Menu *Edycja* zawiera narzędzia wspomagające zarządzanie edycją kodu obliczeń wykonywanych w oknie konsoli (**4**).

Manu *Sterowanie* umożliwia kontrolę wykonywanych przez środowisko poleceń, przerwanie, zatrzymanie i wznowienie.

Kolejne menu *Narzędzia*, zawiera skróty do programów sprzężonych ze środowiskiem Scilab-a, aktywacji lub dezaktywacji okien pomocniczych (**3**, **5** i **6**). Programy sprzężone to:

- *SciNotes* – edytor wspomagający pisanie skryptów dla środowiska Scilab
- *XCOS* – moduł wizualnego programowania metod numerycznych
- *ATOM* – moduł zarządzający dodatkami środowiska Scilab

Ostatnim elementem menu *Narzędzia* jest konwerter projektów Matlab-a do formatu Scilab-a. Ostatnie menu oznaczone znakiem "?" grupuje elementy związane z pomocą systemu Scilab. Zawiera informacje o składni i zastosowaniu poszczególnych poleceń środowiska oraz rozbudowaną bazę przykładów do samodzielnej analizy.

Konsola jest głównym oknem programu Scilab. Umożliwia ona wprowadzanie pojedynczych poleceń, zadań obliczeniowych, otrzymując natychmiast wynik ich działania. Konsola obsługuje wszystkie podstawowe skróty klawiszowe wspomagające pracę z tekstem (*kopiuj*, *wytnij* i *wklej*), szybki dostęp do pomocy. Dodatkową właściwości konsoli jest pamięć wsteczna wprowadzonych poleceń. Dostęp do niej jest realizowany z zastosowanie klawiszy kierunkowych *Up* i *Down*.

1.1. Podstawowe polecenia systemu Scilab

Polecenia Scilab-a można podzielić na pięć podstawowych grup:

- funkcje systemowe – `help`, `clear`, `clc`, `dir` itp.,
- działania matematyczne – dodawania, odejmowania, mnożenia, `sin`, `cos`, `log` itp.,
- rachunek macierzowy – `inv`, `diag`, `max`, itp.,
- metody numeryczne – `Ode`, `solve`, `linsolve` itp.,
- procedury graficzne i tekstowe – `plot2d`, `plot3d`, `disp` itp..

W niniejszym rozdziale omówione zostaną trzy pierwsze grupy poleceń. Pracując w systemie Scilab'a należy pamiętać, że efekt działania każdej linii kodu kończącej się znakiem średnika „`;`" nie będzie wyświetlany na ekranie konsoli. Komentarze w systemie Scilab umieszcza się po podwójnym znaku *slash* „`//`".

1.1.1. Funkcje systemowe

Polecenia z tej grupy umożliwiają zarządzanie pracą konsoli i współdziałaniem z innymi elementami środowiska Scilab.

Jednym z najważniejszych modułów środowiska Scilab-a jest pomoc systemowa „Help". Dostęp do niej jest możliwy poprzez menu pomocy lub skorzystanie z polecenia aktywującego. Możliwe są trzy sposoby aktywacji pomocy z poziomu konsoli:

1. wywołania okna pomocy – `help;`
2. wywołanie okna pomocy z jednoczesnym wyszukaniem informacji o zadanym poleceniu – `help plot3d;`
3. wywołanie okna pomocy z jednoczesnym wyszukaniem informacji na zadany temat – `help („integration");`

Kolejnym narzędziem systemowy konsoli jest polecenie umożliwiające czyszczenie okna konsoli. Polecenie ma dwa tryby pracy:

1. całkowite czyszczenie ekranu konsoli – `clc;`
2. kasowanie wybranej liczby końcowych linii konsoli – `clc(10);`

W środowisku Scilab-a można definiować i korzystać ze zmiennych. Definiowanie polega na przypisaniu do wybranej nazwy wartości. Scilab nie daje możliwości samodzielnego zdefiniowania typu zmiennej, robi to sam w tle analizując daną przypisaną do zmiennej. Wywołanie zmiennej realizuje się poprzez korzystanie z jej symbolicznej nazwy. Scilab rozróżnia przy definiowaniu zmiennych małe i duże litery.

```
a=10;
a
 a  =
    10.
A=20;
```

Do zarządzania istniejącymi zmiennymi służą dwa polecenia:

1. sprawdzanie czy zmienna o podanej nazwie istnieje,

```
A=20;//definiuje zmienną A
B=30;//definiuje zmienną B
isdef("A")//sprawdza czy istnieje zmienna A
 ans  =
  T
//zmienna a istnieje, odpowiedź true T
isdef("b")//sprawdza czy istnieje zmienna b
 ans  =
  F
//zmienna b nie istnieje, odpowiedź false F
```

2. kasowanie zmiennej o zadanej nazwie

```
disp(a,A,B,c)//wyświetlenie zdefiniowanych zmiennych
    12.
    34.
    20.
    10.
clear B//skasowanie zmiennej B
disp(a,A,B,c)//sprawdzenie występowania zmiennych
         !--error 4
Niezdefiniowana zmienna: B
//zmienna B nie istnieje
clear("a","A")//kasowanie zmiennych a i A
disp(a,A,B,c)//sprawdzenie występowania zmiennych
       !--error 4
Niezdefiniowana zmienna: a
//zmienna a nie występuje
c//sprawdzenie zmiennej c
 c  =
    12.
//zmienna A istnieje
A// Sprawdzenie zmiennej A
  !--error 4
Niezdefiniowana zmienna: A
//zmienna A nie występuje
```

Wykorzystane w przykładzie polecenie `disp();` umożliwia wyświetlenie tekstu lub zawartości zmiennych na ekranie konsoli.

Korzystając z procedury `help` należy przeanalizować stosowanie funkcji `disp` i funkcji powiązanych (ze szczególnym uwzględnieniem funkcji `string`), a następnie za jej pomocą wyświetlić zawartość zdefiniowanych zmiennych `a`, `b` i `c` w postaci pokazanej poniżej:

```
Zmienna a wynosi  10
Zmienna b wynosi  16
Zmienna c wynosi  18
```

Kolejną grupę poleceń systemowych Scilab-a stanowią funkcje dyskowe umożliwiających przeprowadzanie operacji na katalogach i plikach. W trakcie zajęć w laboratorium metod numerycznych zastosowanie znajdą funkcje związane z identyfikacją i ustawianiem domyślnego katalogu systemu Scilab-a. Identyfikację aktualnego katalogu roboczego realizuje się poleceniem `pwd`.

```
pwd
 ans  =
  d:\temp
```

Ustawienie nowego katalogu roboczego możne zrealizować dwoma sposobami. Poprzez menu *Plik i* skrót „*Zmiana bieżącego katalogu..*”, lub za pomocą polecenia `chdir`. Wybrany katalog musi istnieć, a jeżeli go nie ma można go utworzyć poleceniem `mkdir`.

```
chdir d:\nowy_temp //definiowanie nowego katalogu
roboczego
 ans  =
  F
//katalog o podanej nazwie nie istnieje
mkdir("d:\nowy_temp") //utworzenie żądanego katalogu
 ans  =
    1.
chdir d:\nowy_temp //definiowanie nowego katalogu
roboczego
 ans  =
  T
// zdefiniowanie potwierdzone
pwd //sprawdzenie poprawności definicji
 ans  =
 d:\nowy_temp
```

Procedury te znajdą zastosowanie w sytuacjach konieczności załadowania funkcji zdefiniowanych w osobnych skryptach zapisanych na dysku.

Ostatnim elementem omawianych poleceń systemowych jest funkcja definiująca format wyświetlania wartości liczbowych. Zadanie to realizowane jest za pomocą funkcji `format`. Funkcja ta posiada dwa parametry sterujące, pierwszy definiujący styl wyświetlania liczby, a drugi ustawiający liczbę znaków stosowanych przy zapisie.

Powszechne zastosowanie mają dwa style zapisu liczby, algebraiczny definiowany za pomocą symbolu `"'v'"` o postaci `123.1234`, oraz potęgowy, o podstawie 10, ustawiany symbolem `"'e'"` o postaci `12.23D+02`. Zwrócić należy uwagę, że gdy w pierwszym przypadku wyświetlana liczba przekroczy zdefiniowany rozmiar, zostanie ona wyświetlona w postaci potęgowej. Ustalając wielkość drugiego parametru należy pamiętać o zarezerwowaniu jednego znaku na znak "-" oraz kropkę "." oddzielającą część ułamkową. W przypadku postaci potęgowej trzy znaki są rezerwowane na zapis potęgi. Minimalna liczba znaków przy zapisie potęgowym wynosi 10, a algebraicznym 2. Jeżeli liczba ustawionego limitu znaków będzie za mała do wyświetlenia liczby to system zapisze jej wartość za pomocą symbolu gwiazdki `"*"`.

```
A=[123456.25,-234567.678;-0.1213344,0.354564335]
 A  =
    123456.25  - 234567.68
```

```
  - 0.1213344    0.3545643
format('v',5) //postać algebraiczna, pięć znaków
A
 A  =
   *****  -*****
  - 0.12    0.35
format('e',15) //postać potęgowa, 15 znaków
A
 A  =
    1.23456250D+05  - 2.34567678D+05
  - 1.21334400D-01    3.54564335D-01
```

1.1.2. Podstawowe działania matematyczne

Scilab jest programem obliczeniowym umożliwiającym wykonywanie działa na liczbach całkowitych, rzeczywistych i zespolonych. Możliwe jest także wykonywanie obliczeń symbolicznych, po doinstalowaniu specjalnego plugin-u, ale zagadnienie to wykracza poza program laboratorium metod numerycznych.

Należy pamiętać, że tak jak i w innych programach obliczeniowych miara kąta definiowana jest w radianach, a ułamki dziesiętne definiuje się za pomocą kropki „.”. Dostępne są jednakże odpowiedniki funkcji trygonometrycznych korzystających z miary kątów w stopniach.

W Scilab-ie zdefiniowano także stałe, takie jak: `%pi` – liczba π, `%e` – liczba e, `%inf` – nieskończoność, oraz `%i` jako operator liczby urojonej w zapisie zespolonym.

Podstawowe działania matematyczne wraz z zastosowaniem nawiasów do doprecyzowania kolejności obliczeń pokazano na przykładzie poniżej.

```
10+5*16/2
 ans  =
    50.
((10+5)*16)/2
 ans  =
    120.
(10+(5*16))/2
 ans  =
    45.
```

Scilab zawiera szereg predefiniowanych funkcji stosowanych w obliczeniach inżynierskich. W tabeli 1.1. zestawiono najważniejsze funkcje matematyczne, wraz z przykładami ilustrującymi ich działanie.

Tabela 1.1 Zestawienie wybranych funkcji predefiniowanych w Scilab-ie

Lp.	Funkcja	Opis	Przykład

1	`sin(x), sind(x)`	Sinus kąta podanego w radianach, stopniach	`sin(%pi/3)`
			` ans  = 0.8660254`
			`sind(60)`
			` ans  = 0.8660254`

2	`cos(x), cosd(x)`	Cosinus kąta podanego w radianach, stopniach	`cos(%pi/3)`
			` ans  = 0.5`
			`cosd(60)`
			` ans  = 0.5`

3	`tan(x), tand(x)`	Tangens kąta podanego w radianach, stopniach	`tan(%pi/3)`
			` ans  = 1.7320508`
			`tand(60)`
			` ans  = 0.7320508`

4	`cotg(x), cotd(x)`	Cotangens kąta podanego w radianach, stopniach	`cotg(%pi/3)`
			` ans  = 0.5773503`
			`cotd(60)`
			` ans  = 0.5773503`

5	`asin(x), asind(x)`	Arcus sinus liczby, wynik w radianach, stopniach	`asin(0.5)`
			` ans  = 0.5235988`
			`asind(0.5)`
			` ans  = 30.`

6	`acos(x), acosd(x)`	Arcus cosinus liczby, wynik w radianach, stopniach	`acos(0.5)`
			` ans  = 1.0471976`
			`acosd(0.5)`
			` ans  = 60.`

7	`atan(x), atand(x)`	Arcus tangens liczby, wynik w radianach, stopniach	`atan(0.25)`
			` ans  = 0.2449787`
			`atand(0.25)`
			` ans  = 14.036243`

8	`acot(x), acotd(x)`	Arcus cotangens liczby, wynik w radianach, stopniach	`acot(0.25)`
			` ans  = 1.3258177`
			`acotd(0.25)`
			` ans  = 75.963757`

9	`sinh(x)`	Sinus hiperboliczny liczby	`sinh(1)` ` ans  = 1.1752012`
10	`asinh(x)`	Arcus sinus hiperboliczny liczby	`asinh(1.1752012)` ` ans  = 1.`

11	`log(x)`	Logarytm naturalny liczby	`log(10)`
			` ans  =`
			`  2.3025851`

12	`log10(x)`	Logarytm dziesiętny liczby	`log10(100)`
			` ans  =`
			`  2.`

13	`sqrt(x)`	Pierwiastek kwadratowy liczby	`sqrt(1256)`
			` ans  =`
			`  35.44009`

14	`exp(x)`	Potęgę wykładniczą liczby e^x	`exp(2)`
			` ans  =`
			`  7.3890561`

Źródło: opracowanie własne

Potęgi i pierwiastki stopni wyższych oblicza się za pomocą operatora „^”, dla liczb rzeczywistych i zespolonych. Pierwiastek liczby liczy się za pomocą zapisu potęgowego.

```
4^5
 ans  =
    1024.
ans^(1/5)
 ans  =
    4.
1024^0.2
 ans  =
    4.
```

Zastosowana w przykładzie zmienna `ans` przyjmuje wartość ostatnio otrzymanego wyniku. Metoda wyznaczania potęg i pierwiastków może także być z powodzeniem stosowana w rachunku liczb zespolonych.

```
z=10+25*%i // definicja liczby zespolonej
 z  =
    10. + 25.i
z^2 //potęga liczby zespolonej z
 ans  =
  - 525. + 500.i
ans^0.5 //pierwiastek kwadratowy liczby zespolonej
 ans  =
    10. + 25.i
```

Scilab zawiera kilka funkcji wspomagających działania na liczbach zespolonych, które zostały zestawione w tabeli 1.2.

Tabela 1.2. Zestawienie funkcji rachunku zespolonego

Lp.	Funkcja	Opis	Przykład

1	`complex(a,b)`	Tworzy liczbę zespoloną na podstawie dwóch liczb rzeczywistych a i b	`complex(23,-45)`
			` ans  =`
			`    23. - 45.i`

2	`conj(Z)`	Tworzy liczbę sprzężoną do wejściowej liczby zespolonej Z	`conj(ans)`
			` ans  =`
			`    23. + 45.i`

3	`israel(Z)`	Sprawdza czy wejściowa liczba jest rzeczywista czy zespolona	`isreal(ans)`
			` ans  =`
			`  F`

4	`real(Z)`	Wyznacza część rzeczywistą liczby zespolonej Z	`real(Z)`
			` ans  =`
			`    23.`

5	`imag(Z)`	Wyznacza część urojoną liczby zespolonej Z	`imag(Z)`
			` ans  =`
			`  - 45.`

Źródło: opracowanie własne

Niestety wśród dostępnych funkcji brakuje umożliwiającej wyznaczenie argumentu liczby zespolonej. Konieczne staje się skorzystanie ze wzoru $\varphi = atg\left(\frac{\mathrm{Im}(Z)}{\mathrm{Re}(Z)}\right)$ wyznaczającego tą wartość.

```
atan(imag(c),real(c)) // argument w radianach
 ans  =
    1.2490458
atand(imag(c),real(c)) // argument w stopniach
 ans  =
    71.565051
```

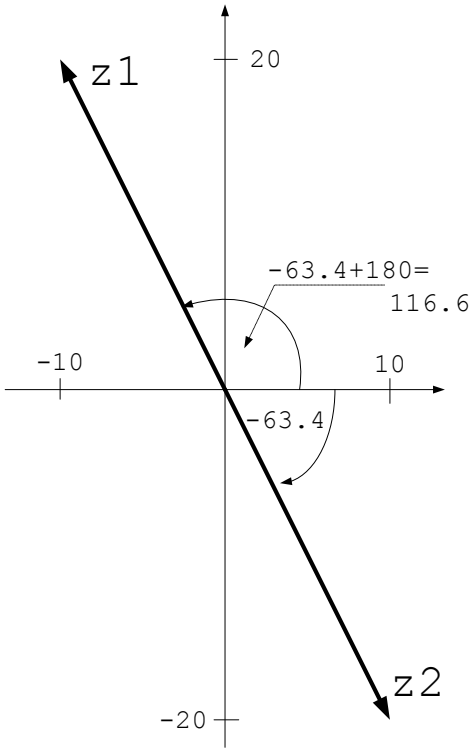
Należy jednak pamiętać, że w sytuacji gdy licznik lub mianownik jest ujemny możemy uzyskać błędny wynik, podobnie jak w sytuacji gdy oba są ujemne. Uzyskana wartość kąta obarczona jest błędem równym 180 stopni.

```
z1=-10+20*%i
 z1  =
   - 10. + 20.i
 z2=10-20*%i
 z2  =
    10. - 20.i
 arg_z1=atand(imag(z1)/real(z1))
 arg_z1  =
   - 63.434949
 arg_z2=atand(imag(z2)/real(z2))
 arg_z2  =
   - 63.434949
```

Uzyskano w obu przypadkach tą samą wartość kąta, przy czym prawidłowo obliczona jest wartość argumentu liczby zespolonej `z2`, co obrazuje rysunek 1.2.

```
arg_z1=arg_z1+180 // korekta argument z1
 arg_z1  =
    116.56505
```

W takiej sytuacji warto napisać własną funkcję realizującą prawidłowe obliczanie argumentu liczby zespolonej. Zagadnienia programowania w środowisku Scilab omówione zostaną w rozdziale 2.



Rys. 1.2. Obrazowanie graficzne określania poprawnego wyniku obliczania argumentu liczby zespolonej
Źródło: opracowanie własne

1.1.3. Rachunek macierzowy

Scilab jest narzędziem bardzo dobrze wyposażonym do przeprowadzania obliczeń w oparciu o rachunek macierzowy. Podobnie jak w przypadku omówionych wcześniej zmiennych, w sposób analogiczny można definiować macierze. Scilab zmienną liczbową traktuje jak macierz jednoelementową. Wektory są macierzami o jednej kolumnie lub jednym wierszu.

```
A=[1,2,3]
 A  =
    1.    2.    3.
B=[1;2;3]
 B  =
    1.
    2.
    3.
```

```
C=[1,2,3;4,5,6;7,8,9]
 C  =
    1.    2.    3.
    4.    5.    6.
    7.    8.    9.
```

W pokazanym powyżej przykładzie zdefiniowano trzy zmienne macierzowe, pierwsze dwie można określić jako wektory. Elementy występujące w jednym wierszu oddzielane są przecinkiem ",", z kolei wiersze oddzielane są znakiem średnika ";". Macierze zawierające elementy ciągów arytmetycznych lub geometrycznych można generować za pomocą poleceń systemowych, bez konieczności ręcznego wpisywania poszczególnych elementów, co jest szczególnie uciążliwe przy tworzeniu dużych macierzy.

```
A=[0:2.3:10]
 A  =
    0.    2.3    4.6    6.9    9.2
```

Definicja ciągu składa się z trzech parametrów, pierwszy parametr określa pierwszy element ciągu, drugi przyrost pomiędzy kolejnymi elementami ciągu, a ostatni określa wartość maksymalną. Przy czym końcowy element ciągu wcale nie musi być równy wartości maksymalnej, nie może być tylko od niej większy. Drugi parametr nie jest obowiązkowy. Jego pominięcie oznacza, że przyrost będzie wynosił 1.

Podobnie jak wektor można definiować macierz, należy jednak pamiętać, że za pomocą ciągu można definiować tylko elementy wierszy, natomiast liczba kolumn w każdym wierszu musi być jednakowa.

```
[1:5;3:7;5:9]
 ans  =
    1.    2.    3.    4.    5.
    3.    4.    5.    6.    7.
    5.    6.    7.    8.    9.
```

W sytuacji gdy chcemy uzyskać ciągi w elementach kolumn, można utworzyć macierz zawierającą oczekiwane elementy w wierszach, a następnie ją transponować uzyskując macierz zawierającą ciągi w elementach kolumn. Transponowanie macierzy realizuje się za pomocą operatora apostrofu "'". Należy jednak pamiętać, że w przypadku transponowania macierzy zawierającej elementy zespolone uzyskamy w kolumnach wartości sprzężone w stosunku do warności jakie były w wierszach macierzy bazowej. W takiej sytuacji należy zastosować operator rachunku skalarnego modyfikując omówiony wcześniej operator do postaci ".'".

```
 A=[1+%i,2,-3*%i;%i,-3,-3+%i;1,4,5*%i]
 A  =
    1. + i       2.  - 3.i
    i          - 3.  - 3. + i
    1.           4.    5.i
A'
 ans  =
    1. - i     - i            1.
    2.         - 3.           4.
    3.i        - 3. - i     - 5.i
A.'
 ans  =
    1. + i       i            1.
    2.         - 3.           4.
  - 3.i        - 3. + i       5.i
```

Do generowania ciągów dostępne są w Scilab-ie jeszcze dwie funkcje `linspace i logspace`. Pierwsza definiuje ciąg liniowy a druga logarytmiczny. Każda z nich definiowana jest przez trzy parametry.

Dla ciągu liniowego pierwszy i drugi parametr definiują początek i koniec ciągu, a trzeci liczbę elementów występujących w ciągu.

```
linspace(0,10,5)
 ans  =
    0.    2.5    5.    7.5    10.
```

Za pomocą kombinacji poleceń `linspace` także można wygenerować macierz uzyskując liniowe rozkłady wartości liczbowych w poszczególnych wierszach.

```
[linspace(0,10,5);linspace(0,25,5);1,2,3,4,5]
 ans  =
    0.    2.5     5.      7.5      10.
    0.    6.25    12.5    18.75    25.
    1.    2.      3.      4.       5.
```

Podobnie wygląda generowanie ciągu logarytmicznego, o podstawie 10. Pierwszy i drugi parametr definiują początek i koniec ciągu poprzez podanie wykładnika potęgi, trzeci parametr określa liczbę elementów ciągu

```
logspace(0,4,4)
 ans  =
    1.    21.544347    464.15888    10000.
```

Scilab posiada kilka funkcji tworzących macierze charakterystyczne, takie jak diagonalna, jednostkowa, jedynkową, zerową i losową. W przypadku macierzy diagonalnej jako parametr funkcji należy podać wektor elementów znajdujących

się na przekątnej macierzy. W pozostałych przypadkach podaje się tylko wymiar macierzy. Generator liczb losowych działa w przedziale <0,1>.

```
diag(logspace(0,2,5)) // macierz diagonalna
 ans  =
    1.    0.          0.     0.          0.
    0.    3.1622777   0.     0.          0.
    0.    0.          10.    0.          0.
    0.    0.          0.     31.622777   0.
    0.    0.          0.     0.          100.

eye(4,3) // macierz jednostkowa
 ans  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.
    0.    0.    0.

ones(4,3) // macierz jedynkowa
 ans  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.

zeros(3,3) // macierz zerowa
 ans  =
    0.    0.    0.
    0.    0.    0.
    0.    0.    0.

rand(4,3) // macierz losowa
 ans  =
    0.1100157    0.6359214    0.7989802
    0.3777934    0.8685321    0.2169637
    0.0348341    0.0725780    0.196027
    0.8525960    0.5971688    0.6935654
```

Odwołanie się do wybranego elementu macierzy realizowane jest poprzez podanie współrzędnych wybranego elementu, pierwszy parametr to numer wiersza, a drugi to numer kolumny.

```
ans(2,3)
 ans  =
   0.2169637
```

Kolejnym parametrem wspomagającym pracę z macierzami jest symbol ostatniego elementu wiersza lub kolumny "$". Za jego pomocą można odwołać się do wybranych elementów macierzy względem jej końca.

```
A=rand(5,3)
 A  =
    0.7666675    0.6783865    0.8678611
    0.2493243    0.4896997    0.0944029
    0.2224035    0.0201628    0.8563716
    0.8650801    0.8276945    0.4521064
    0.7313460    0.1784218    0.4435688
B=A($-1:$,$-1:$) //odwołanie do przedziału
 B  =
    0.8276945    0.4521064
    0.1784218    0.4435688
C=A($,$-2) //odwołanie do wartości
 C  =
    0.7313460
```

W pewnych sytuacjach może być konieczne wyszukanie największej i najmniejszej wartości zawartej w macierzy. Do tej czynności służą polecenia `max` i `min`.

```
max(A)
 ans  =
    0.8678611
min(A)
 ans  =
    0.0201628
```

Wykonując działania matematyczne w rachunku macierzowym należy pamiętać, że muszą być spełnione określone warunki, aby można było wykonać określone działania.

Dodawanie – jest działaniem przemiennym i macierze muszą mieć jednakowy wymiar.

Odejmowanie – macierze muszą mieć jednakowy wymiar.

Mnożenie – nie jest działaniem przemiennym, liczba kolumn pierwszego czynnika musi być równa liczbie wierszy drugiego czynnika

Dzielenie – obydwa czynniki muszą mieć jednakową liczbę kolumn.

```
A
 A  =
    0.1370117    0.6787988    0.4428428
    0.8982042    0.8514022    0.4394386
```

```
B
 B  =
    0.8925602    0.683989     0.5557416
    0.0964628    0.8480543    0.6402073
    0.1351030    0.6676246    0.1847532
    0.0352035    0.7631582    0.4449726
C
 C  =
    0.0563957    0.9367536
    0.3927477    0.0739401
A+B
    !--error 8 Niezgodne dodawanie.
A*B
    !--error 10 Niezgodne mnożenie.
A*C
    !--error 10 Niezgodne mnożenie.
C*A
 ans  =
    0.8491229    0.8358354    0.4366201
    0.1202243    0.3295494    0.2064176
B/C
    !--error 266
A i B muszą mieć taką samą ilość kolumn.
B/A
 ans  =
  - 0.0607879    0.9756164
    1.4548158  - 0.1234483
    0.7273646    0.0693017
    1.2562143  - 0.1438747
A/B
 ans  =
    0.0570570    0.5772497    0.2250267    0.
    0.9289909  - 0.3368724    0.7514235    0.
```

Scilab zawiera także funkcje wykonujące bardziej złożone obliczenia i przekształcenia macierzy. We wszystkich jedynym parametrem jest macierz lub zmienna jej odpowiadająca. Wymienić to należy funkcję liczącą macierz odwrotną, oraz wyznaczającą wartość bezwzględną, lub w przypadku liczb zespolonych, długość wektora. W przypadku macierzy odwrotnej można skorzystać także z operatora potęgi.

```
F=[1+%i,1;-1+%i,-%i]
 F  =
    1. + i       1.
  - 1. + i     - i
inv(F)
 ans  =
```

```
    0.25 - 0.25i  - 0.25 - 0.25i
    0.5             0.5i
F^-1
 ans  =
    0.25 - 0.25i  - 0.25 - 0.25i
    0.5             0.5i
abs(F)
 ans  =
    1.4142136    1.
    1.4142136    1.
```

Zadania

W oparciu o materiał wykładowy omówiony w powyższym rozdziale należy wykonać ćwiczenia zestawione poniżej.

1. Korzystając z pomocy systemu Scilab przeanalizować dostępne informacje o omówionych w ćwiczeniu poleceniach.
2. Przed przystąpieniem do wykonywania ćwiczeń wyczyścić ekran konsoli oraz pamięć systemową z istniejących zmiennych, ustawić domyślny katalog na swój katalog roboczy.
3. Utworzyć trzy zmienne:
   a) macierz o wymiarze 3x4 zawierającą losowo wygenerowane liczby z zakresu <-10,10>,
   b) macierz o wymiarze 4x2 zawierającą losowo wygenerowane liczby z zakresu <-2,10>,
   c) macierz o wymiarze 5x4 zawierającą w kolumnach liczby w ciągu logarytmicznym w zakresie <2,200>.
4. Utworzyć dwie zmienne zespolone, w wartościach wygenerowanych losowo:
   a) wektor o długości 5,
   b) macierz o wymiarze 3x5.
5. Zmienić format wyświetlania liczb na potęgowy, format 11 znaków, a następnie na algebraiczny zdefiniowany przez 6 znaków. Wyświetlić zmienne w obu formatach.
6. W oparciu o zdefiniowane zmienne przetestować omówione w ćwiczeniu działania i funkcje

2. Programowanie Scilab-a

Podstawowym sposobem efektywnego wykorzystania Scilab-a jest programowanie obliczeń do wykonania. Pisanie programów ma szereg zalet w stosunku do bezpośredniej pracy na konsoli programu:

1) wykonanie nieraz bardzo skomplikowanych obliczeń sprowadza się do prostego wywołania odpowiedniego programu,
2) zauważone błędy jest łatwiej poprawiać poddając edycji treść kodu źródłowego,
3) zawsze zostaje na dysku komputera coś co można zabrać ze sobą, udoskonalać i przerabiać.

Pliki wykonalne Scilab-a (skrypty obliczeniowe) są zwykłymi plikami tekstowymi, które można przygotować za pomocą dowolnego edytora tekstu. Warto jednak pamiętać, że środowisko Scilab zostało zaopatrzone w dobry edytor tekstu o nazwie SciNotes (dawniej SciPad). Do zalet tego narzędzia programisty należą: kolorowanie kodu, automatyczne uzupełnienie typowych elementów programu oraz współpraca z programem obliczeniowym. Mimo zwykłej zawartości tekstowej, skrypty Scilab-a powinny być zapisywane jako pliki dyskowe o rozszerzeniu nazwy sce lub sci.

Wykonanie zaprogramowanych w skrypcie obliczeń może być uruchomione za pomocą polecenia **exec**, odpowiedniego skrótu klawiaturowego edytora SciNotes lub polecenia z menu **Wykonaj** tego programu.

Najprostszy skrypt o nazwie skr1.sce i podanej poniżej zawartości został zapisany w katalogu D:\Student.

```
// skasowanie zmiennych z pamięci środowiska
clear
// przypisanie wartości zmiennych
x = 2;
y = 3;
// obliczenia:)
z = x * y;
// wyświetlenie wartości zmiennej
// nie ma średnika na końcu takiego wiersza
z
```

W powyższym skrypcie zostały umieszczone komentarze programisty. Tekst rozpoczynający się podwójnym ukośnikiem nie jest analizowany i wykonywany. Stanowi opis wyjaśniający budowę lub działanie skryptu obliczeniowego i jest niezwykle ważnym zbiorem informacji dla każdego kto zamierza analizować, wykorzystywać lub rozbudowywać program w przyszłości.

Wykonanie skryptu za pomocą polecenie **exec** wymaga podania pełnej ścieżki dostępu do pliku ze skryptem lub zmiany katalogu roboczego za pomocą polecenia **cd**. Poniższe polecenia wykonano w konsoli Scilab-a:

```
cd 'D:\Student'
 ans  =
 D:\Student
exec 'skr1.sce'
clear
x = 2;
y = 3;
z
 ans  =
    6.
```

Po zmianie katalogu roboczego i wywołaniu skryptu wszystkie jego instrukcje zostały potraktowane tak jakby zostały wprowadzone na klawiaturze – widać znaki zachęty, treść każdego polecenia i komentarze. Mimo, że ostatecznie spodziewany wynik obliczeń pojawia się jako wartość zmiennej **ans**, to powyższy sposób wywoływania skryptów nie należy chyba do najbardziej eleganckich. Można zrobić to inaczej:

```
exec('D:\Student\skr1.sce', 0)
 ans  =
    6.
```

W tym przypadku polecenie **exec** zostało uruchomione z podaniem pełnej ścieżki dostępu do skryptu oraz oznaczeniem trybu wyświetlania. Wartość 0 (domyślna) powoduje, że wyświetlane są skutki działania tylko tych linii skryptu, które nie zostały zakończone średnikiem.

Tabela 2.1. Podstawowe tryby polecenia exec

nr trybu	**działanie**
-1	bez wyświetlania
0 (domyślny)	wyświetlane są wyniki
1	wyświetlane są znaki zachęty, instrukcje i wyniki
7	wykonanie skryptu krok po kroku z wyświetlaniem instrukcji i wyników

Źródło: opracowanie własne na podstawie [9]

Skrypt może być uruchamiany bezpośrednio z edytora SciNotes przy wykorzystaniu skrótów klawiaturowych lub poleceń z menu **Wykonaj**.

Tabela 2.2. Uruchamianie skryptów z programu SciNotes

menu Wykonaj	skrót klawiaturowy	tryb polecenia exec
plik, bez echa	Ctrl + Shift +E	-1
plik, z echem	Ctrl + E	1
do kursora, z echem	Ctrl + L	0
Zapisz i wykonaj	F5	-1

Źródło: opracowanie własne na podstawie [9]

2.1. Instrukcja warunkowa if

W każdym języku programowania istnieje możliwość warunkowego wykonania części programu. Decyzja podejmowana jest na podstawie wyrażenia logicznego, które może przyjmować wartość prawda lub fałsz.

Przygotowano do wykonania poniższy skrypt:

```
a = input('Podaj liczbę: ');
if a > 0 then
    disp('liczba większa od zera');
end
```

Po uruchomieniu użytkownik jest zachęcany do podania wartości liczbowej, która za pomocą polecenia **input** trafia z klawiatury do zmiennej *a*. Na podstawie tej wartości podejmowana jest w tym przypadku najprostsza decyzja: jeśli warunek logiczny a > 0 jest prawdziwy, zostaje wyświetlony komunikat. Jeśli warunek nie jest spełniony – nic się nie stanie.

Powyższy przykład można rozszerzyć. Najczęściej instrukcja warunkowa jest programowana w następującej postaci:

```
if a > 0 then
    disp('liczba większa od zera');
else
    disp('liczba mniejsza od zera lub zero');
end
```

Po słowie kluczowym **else** można więc podać polecenia wykonywane w sytuacji gdy warunek logiczny daje wartość fałszywą.

Ostatnią formę instrukcji warunkowej **if** objaśnia następujący przykład:

```
if a > 0 then
    disp('liczba większa od zera');
elseif a == 0 then
    disp('liczba zero');
```

```
else
    disp('liczba mniejsza od zera');
end
```

Słowo kluczowe **elseif** z kolejnym warunkiem logicznym może być wykorzystywane wielokrotnie.

Należy zauważyć, że w ostatnim zapisie występuje dwa warunki logiczne: pierwszy a > 0 i drugi a == 0 czyli sprawdzenie czy a ma wartość zero (tabela 2.3). Podwójny znak równości pozwala na tworzenie warunków logicznych, pojedynczy znak równości stanowi operację przypisania wartości – nie można ich mylić. Złożone warunki logiczne można budować w oparciu o operatory logiczne (tabela 2.4).

Scilab nie wymaga zapisu słowa kluczowego **then** ale jeśli już zostanie użyte to instrukcja warunkowa przypomina konstrukcje używane w innych językach programowania.

Tabela 2.3. Operatory relacyjne

operator	opis
==	równe
<> lub ~=	nierówne
>	większe
<	mniejsze
<=	mniejsze lub równe
>=	większe lub równe

Źródło: opracowanie własne na podstawie [9]

Tabela 2.4. Operatory logiczne

operator	opis
&	iloczyn (and)
|	suma (or)
~	negacja (not)

Źródło: opracowanie własne na podstawie [9]

2.2. Instrukcja warunkowa select

Pisanie wielu warunków typu **elseif** może być w niektórych sytuacjach mało efektywne. Możliwe jest natomiast zastosowanie instrukcji, która pozwoli na analizę wartości wskazanej zmiennej. Poniższy przykład jest bardzo prosty ale wystarczy wyobrazić sobie, że liter do wyboru będzie np. dziesięć aby docenić tę konstrukcję.

```
s = input('Wybierz A lub B: ','string');
s = convstr(s, 'u')
select s
case 'A' then
    disp('Wybrano polecenie A')
case 'B' then
    disp('Wybrano polecenie B')
else
    disp('Wybrano niewłaściwe polecenie')
end
```

Użytkownik ma podać z klawiatury jedną literę oznaczającą na przykład pozycję w menu programu obliczeniowego. Tym razem polecenie **input** zostało wykorzystane do wczytania łańcucha tekstowego (modyfikator 'string'). Użytkownik podaje małą lub wielką literę w zależności od humoru. Program jest na to przygotowany i za pomocą instrukcji konwersji łańcuchów tekstowych **convstr** zawsze zamienia odpowiedź na wielkie litery (u – upper case). Ostatecznie wartość zmiennej *s* jest analizowana instrukcją **select**. Każdy przewidziany przypadek ma swój wpis w postaci pary **case** – **then**, które muszą być zapisane w pojedynczej linii skryptu. Jeśli wartość zmiennej jest równa wskazanej po słowie **case** wartości, wykonywany jest właściwy blok instrukcji (w tym przypadku tylko **disp**). Jeśli zmienna *s* zawiera nieprzewidzianą wartość wtedy zadziała warunek **else** i jego ostatnia grupa instrukcji.

2.3. Pętle

Powtarzanie wykonania pojedynczej instrukcji lub grupy instrukcji może być zorganizowane za pomocą konstrukcji programistycznej nazywanej pętlą.

2.3.1. Pętla for

Podstawowym rodzajem pętli jest instrukcja **for**. Liczba powtórzeń zależy od zadanego zbioru wartości iteratora (zmiennej licznikowej pętli). Działanie tej pętli obrazuje następujący przykład:

```
for i = 1:4
    disp(i);
end
```

Po uruchomieniu powyższego skryptu wyświetlone są liczby naturalne od 1 do 4. Instrukcja wewnętrzna pętli (ciało pętli) została wykonana dla każdej wartości iteratora (zmienna *i*), a ten przyjmował wartości z podanego zakresu 1:4. Pamiętając, że w Scilab-ie można na różne sposoby definiować listy wartości, poniższy skrypt będzie wyliczać wartości całkowite od 10 w kierunku malejącym do 0.

```
for i = 10:-1:0
    disp(i);
end
```

Scilab nie zmusza programisty aby wartości liczbowe iteratora były całkowite. Poniższy skrypt wyświetli wartości od 1 do 2 z krokiem 0,1.

```
for i = linspace(1,2,11)
    disp(i);
end
```

Możliwe jest zadanie dowolnej listy wartości dla iteratora. Mogą być to także łańcuchy tekstowe.

```
for i = ['raz' 'dwa' 'trzy']
    disp(i);
end
```

2.3.2. Pętla while

Instrukcje wewnętrzne pętli **while** są uruchamiane jeśli warunek logiczny sprawdzany przed pierwszym i każdym następnym wykonaniem jest prawdziwy. Poniższy skrypt wyświetli więc kwadraty liczb od 1 do 6.

```
i = 1;
while i < 7
    disp(i^2);
    i = i + 1;
end
```

Należy zwrócić uwagę na to, aby warunek logiczny ostatecznie dał wartość fałszywą. Bez tego pętla mogłaby być wykonywana w nieskończoność.

Dopuszczalne są inne formy zapisu tej pętli z użyciem słów kluczowych **do** oraz **then**:

```
i = 1;
while i < 7 do
    disp(i^3); // tym razem do potęgi trzeciej
    i = i + 1;
end
i = 1;
while i < 7 then
    disp(sqrt(i)); // tym razem pierwiastek kwadratowy
    i = i + 1;
end
```

Może się zdarzyć, że z powodu uzyskanych wyników, obliczenia w pętli nie muszą być kontynuowane ponieważ prowadziłoby to do błędów lub było stratą czasu. Działanie pętli można przerwać za pomocą instrukcji **break.** Komenda ta spowoduje, że wykonana będzie kolejna czyli pierwsza po pętli instrukcja skryptu. Poniższy skrypt sprawdza chyba w najmniej efektywny i zasługujący na krytykę sposób czy podana z klawiatury liczba jest liczbą pierwszą.

```
clear
x = input('Podaj liczbę naturalną większą od 1: ');
i = 2;
while i < x
    if modulo(x, i) == 0 break; end
    i = i + 1;
end
if i == x then
    disp('To liczba pierwsza')
end
```

Pętla **while** odpowiada za szukanie w zakresie od 2 do *x*-1 dzielników podanej liczby *x*. Funkcja **modulo** oblicza resztę z dzielenia *x* przez aktualną wartość iteratora *i*. Jeżeli reszta wynosi zero oznacza to, że znaleziono dzielnik liczby *x*, a szukanie dalszych dzielników traci sens. Stąd użycie instrukcji **break**.

2.4. Funkcje użytkownika

„Dziel i rządź" (łac. divide et impera) to nie tylko stara zasada sprawowania rządów. Jej odmiana „dziel i zwyciężaj" (ang. divide and conquer) to świetna metoda realizacji zagadnień programistycznych polegająca na rozbijaniu dużych problemów na tak małe, aby stały się możliwe do rozwiązania. Makra, procedury, funkcje to środki służące do urzeczywistnienia tego pomysłu.

Funkcje Scilab-a to podprogramy definiowane przez użytkownika zdolne do wykonywania wyodrębnionych zadań obliczeniowych na podstawie przekazanych z zewnątrz parametrów. Funkcje te mogą być zapisywane w oddzielnych plikach ładowanych później poleceniem exec (np. biblioteki funkcji), mogą być też częścią skryptu użytkownika.

```
// załadowanie funkcji z pliku dyskowego
exec SCI/modules/elementary_functions/macros/log10.sci;

// definicja funkcji pierwiastek
function w = pierwiastek(x)
    w = sqrt(x);
endfunction

// definicja funkcji srednie
```

```
function [a, g] = srednie(x, y)
    a = (x + y)/2;
    g = sqrt(x*y);
endfunction

//definicja funkcji kw_sz
deff('[k,s]=kw_sz(x)',['k=x*x; s=x*x*x'])

// wywołania funkcji
x = log10(100);
p2 = pierwiastek(2);
[arytm, geometr] = srednie(2, 3);
[kwad, szesc] = kw_sz(4);
```

Funkcje są definiowane na dwa sposoby: za pomocą konstrukcji ze słowami kluczowymi **function** – **endfunction** lub polecenia **deff.** Pierwszy sposób jest bardziej przejrzysty – używany przy pisaniu skryptów, drugi bardziej zwarty – stosowany do definiowania funkcji bezpośrednio na konsoli Scilab-a. W każdym przypadku należy podać nagłówek funkcji zawierający listę zmiennych wynikowych, unikalną nazwę funkcji i listę argumentów jej wywołania, a następnie ciało funkcji.

Zmienne wykorzystywane w programie można podzielić na globalne i lokalne. Zmienne globalne są definiowane w głównym programie, zmienne lokalne – wewnątrz funkcji. Zmienne lokalne kończą swój „żywot" w momencie zakończenia wywołania danej funkcji (powrotu do bloku wywołującego).

Jeśli w funkcji użyto zmiennej, która nie została zdefiniowana wewnątrz funkcji i nie znajduje się na liście parametrów wywołania funkcji to jej wartość jest pobierana z bloku wywołującego pod warunkiem, że zmienna o takiej nazwie tam istnieje.

2.5. Podstawowe operacje wejścia-wyjścia

Scilab umożliwia zapisywanie wartości zmiennych do pliku dyskowego o zawartości binarnej za pomocą polecenia **save**. Jeśli nie zostanie użyta pełna ścieżka dostępu to zapis odbędzie się w aktualnym katalogu roboczym Scilab-a:

```
cd 'D:\Student'
 ans  =  D:\Student
A = rand(10, 10);
B = eye(10, 10);
save('zmaib.dat', 'A', 'B')
```

Ponowne wczytanie zmiennych do środowiska obliczeniowego wykonuje się poleceniem **load**:

```
clear
B
  !--error 4
Niezdefiniowana zmienna: B
load('zmaib.dat')
B(10,10)
ans  =
    1.
```

Zapis wartości zmiennej do pliku w formacie czytelnym dla człowieka można wykonać za pomocą polecenia **print**:

```
print('zmb.txt', B)
```

W praktyce często zachodzi potrzeba, aby duże zbiory danych liczbowych zgromadzonych np. za pomocą cyfrowego sprzętu pomiarowego wprowadzić do środowiska obliczeniowego w celu analizy i wizualizacji. Pomocny okazuje się import danych z plików dyskowych w formacie csv (ang. Comma Separated Values).

Plik dyskowy o nazwie dane.csv umieszczony w katalogu roboczym Scilab-a zawiera rekordy składające się z dwóch pól reprezentujących czas pomiaru i wartość temperatury:

```
   4.000E-03   7.794E+01
   4.100E-03   7.815E+01
   4.200E-03   7.839E+01
...
   5.900E-03   8.748E+01
```

Należy zauważyć, że nie jest to poprawnie sformatowany plik typu csv ponieważ separatorem pól rekordu jest ciąg trzech spacji z czego wynika, że każdy rekord ma trzy pola i pierwsze pole każdego rekordu jest puste.

Wczytanie zawartości pliku do Scilab-a wykonuje się za pomocą polecenia **csvRead** ale przygotowanego na nietypowy separator:

```
M = csvRead('dane.csv', '   ')
 M =
    Nan    0.004     77.94
    Nan    0.0041    78.15
    Nan    0.0042    78.39
...
```

Pierwszym argumentem funkcji **csvRead** była nazwa pliku, drugim – separator pól rekordu składający się z trzech spacji. Wczytane zostały rekordy składające się z trzech pól przy czym pierwsze pole nie zawiera poprawnej wartości liczbowej (ang. Nan – not a numer).

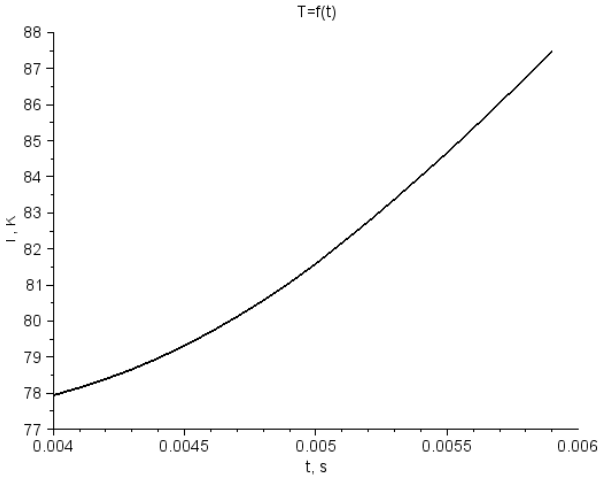
Można zrobić to lepiej:

```
M = csvRead('dane.csv','   ', [], [], [], [], [1 2 100 3])
 M  =
    0.004     77.94
    0.0041    78.15
    0.0042    78.39
...
```

Tym razem wczytane zostały wskazane kolumny danych (i rekordy) pliku. Parametr w postaci macierzy [1 2 100 3] oznacza wczytanie od 1. do 100. rekordu, od 2. do 3. kolumny. Puste nawiasy klamrowe oznaczają inne niewykorzystane tym razem opcje funkcji **csvRead.**

Można teraz wykonać np. wykres pozyskanych danych:

```
plot2d(M(:,1), M(:,2))
xtitle('T=f(t)', 't, s', 'T, K')
```



Wykres 2.1. Przebieg temperatury w funkcji czasu wykreślony na podstawie danych liczbowych z pliku typu csv
Źródło: opracowanie własne

Do zapisu danych do pliku w formacie csv służy polecenie **csvWrite.** Podstawowe wywołanie wymaga jedynie podania zmiennej (macierzy) zawierającej dane do zapisu oraz nazwy pliku dyskowego:

```
t = linspace (0, 20e-3, 101);
u = 100 * sin(2 * %pi * 50 * t);
csvWrite([t' u'], 'wyniki.csv')
```

W pliku o nazwie wyniki.csv zapisano rekordy składające się z dwóch pól: czasu i wartości chwilowych napięcia sinusoidalnego o amplitudzie 100 V, częstotliwości 50 Hz i zerowej fazie początkowej. Domyślnym separatorem pól jest przecinek, znakiem dziesiętnym kropka. Zapis `[t' f']` posłużył do utworzenia macierzy o dwóch kolumnach na bazie transponowanych macierzy wierszowych czasu i napięcia.

2.6. Operacje na łańcuchach tekstowych

Scilab umożliwia zapisywanie łańcuchów tekstowych z użyciem apostrofów lub cudzysłowu.

Konkatenacja (łączenie) łańcuchów:

```
'raz' + 'dwa' + 'trzy'
 ans  =
 razdwatrzy
```

Obliczanie długości łańcucha:

```
s = '12345678901234567890';

length(s)
 ans  =
    20.
```

Kopiowanie fragmentów łańcucha:

```
part(s, 12)
 ans  =
 2
part(s, 12:17)
 ans  =
 234567
part(s, [11, 15, 19])
 ans  =
 159
```

Szukanie w łańcuchu:

```
strindex(s, '89')
 ans  =
    8.    18.
```

Zastępowanie fragmentu:

```
strsubst(s, '123', 'xxx')
 ans  =
 xxx4567890xxx4567890
```

W większości przypadków Scilab wykonuje konwersje liczbowo-tekstowe i tekstowo-liczbowe „w locie”. Jednak w szczególnych sytuacjach może być potrzebna jawna zamiana wartości liczbowej na łańcuch tekstowy:

```
x=13/11;
'Wynik wynosi x = ' + string(x)
 ans  =
 Wynik wynosi x = 1.1818182
```

Utworzenie wektora łańcuchów tekstowych:

```
ws = string(10:20)
 ws  =
!10  11  12  13  14  15  16  17  18  19  20  !

ws(5)
 ans  =
 14
```

Obliczanie wartości (ewaluacja) łańcucha tekstowego:

```
a = 1; b = 2;

M = ['a', 'b']
 M  =
!a  b  !

x = evstr(M)
 x  =
    1.    2.
```

Zadania

1. Napisać skrypt obliczający pierwiastki równania kwadratowego o współczynnikach podanych przez użytkownika.
2. Napisać skrypt obliczający odchylenie standardowe dla podanego przez użytkownika zbioru wartości liczbowych.

Napisać skrypt obliczający wartość funkcji:

$$f(x)=\begin{cases}\ln(x) & 0<x\le 1\\ \sqrt[3]{8(x-1)} & 1<x\le 2\\ 0 & x\le 0\vee x>2\end{cases}$$

3. Napisać skrypt do iteracyjnego obliczania silni podanej przez użytkownika liczby.
4. Napisać skrypt do rekurencyjnego obliczania silni podanej przez użytkownika liczby. Zdefiniować odpowiednią funkcję.
5. Napisać skrypt sprawdzający czy podana przez użytkownika liczba jest liczbą pierwszą.
6. Używając funkcji użytkownika napisać skrypt obliczający kolejne liczby doskonałe. Liczba doskonała to taka liczba naturalna, która jest sumą wszystkich swoich dzielników właściwych. Dzielnik właściwy to każdy dzielnik mniejszy od tej liczby. Przykładem takich liczb są 6, 28, 496.

   D6={1,2,3} i 1+2+3 = 6,
   D28={1,2,4,7,14} i 1+2+4+7+14 = 28
   D496={1,2,4,8,16,31,62,124,248} i 1+2+4+8+16+31+62+124+248 = 496

Podpowiedź: parzyste liczby doskonałe mają postać: $n=2^{k-1}(2^k-1)$, o ile 2^k-1 jest liczbą pierwszą (k - pewna liczba naturalna). Twierdzenie powyższe udowodnił Euklides.

3. Grafika w środowisku Scilab

Najistotniejszym elementem każdego dokumentu prezentującego wyniki badań pomiarowych lub symulacyjnych są właściwie przygotowane wykresy. Powinny one prezentować dane w sposób jasny i czytelny.

Scilab posiada rozbudowany aparat narzędziowy przeznaczony do definiowania i prezentacji wykresów wraz z ich opisem. Możliwe jest tworzenie wykresów dwuwymiarowych i trójwymiarowych [3].

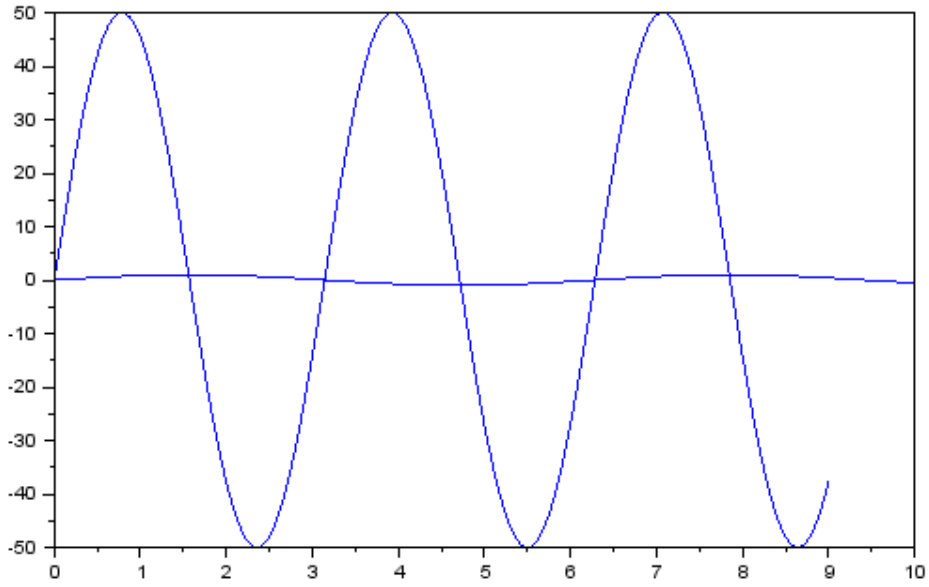
Grafika w środowisku Scilab to nie tylko wykresy, ale także graficzny interfejs użytkownika GUI, opracowany model numeryczny i prezentację wyników obliczeń – wykresy. Osobną grupę stanowią polecenia zarządzające wyświetlaniem okna graficznego oraz wyświetlanego w nim tekstu.

Przy tworzeniu skryptu realizującego obliczenia i prezentującego wyniki w postaci liczbowej i graficznej na początku należy przeprowadzić procedurę czyszczenia pamięci, ekranu konsoli i okna graficznego. Czynność tę należy wykonać w celu uniknięcia problemów wynikających z danych pozostawionych w systemie po wcześniej wykonanych symulacjach - obliczeniach.

```
clear; //czyszczenie pamięci ze zmiennych
clc; //czyszczenie konsoli
clf; // czyści i przywraca do ustawień standardowych
   aktywne okno graficzne – okno graficzne pozostaje
xdel(winsid()); //zamknięcie wszystkich okien graficznych
```

Zastosowanie polecenia `xdel` bez żadnych parametrów spowoduje wyczyszczenie aktywnego okna graficznego [1]. Czyszczenie okienek graficznych należy stosować ze względu na procedurę rysowania wykresu. W przypadku nie wyczyszczenia istniejącego okna graficznego przed narysowaniem nowego wykresu w nim spowoduje że system narysuje nowy wykres na istniejącym. W efekcie będą widoczne dwa nakładające się wykresy, co może prowadzić do uzyskania nieczytelnych i nieprawidłowo wykreślonych przebiegów, jak pokazano na wykresie 3.1.

```
clear;clc;clf;
xdel(winsid());
t=0:0.01:10;
u=sin(t);
plot(t,u);
t=0:0.01:9
i=100*sin(t).*cos(t);
plot(t,i);
```



Wykres 3.1. Nałożenie dwóch wykresów na siebie
Źródło: opracowanie własne

W przypadku konieczności wykonania kilku wykresów można przyjąć jedną z trzech metod:

1. o ile warunki pozwalają przedstawić je na wspólnym wykresie,
2. podzielić okno graficzne na części, aby każdy z wykresów był narysowany osobno,
3. utworzyć nowe okno graficzne dla każdego wykresu.

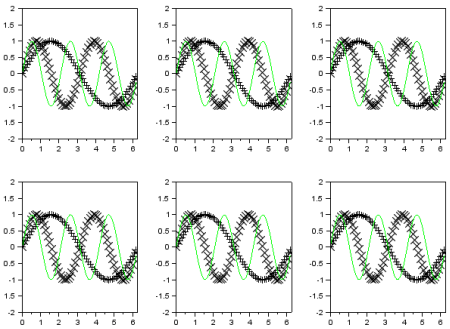
Pierwsza opcja realizowana jest przez nałożenie na siebie dwóch wykresów lub wykorzystanie właściwości procedury rysującej wykres. Zagadnienie to zostanie omówione w kolejnych rozdziałach.

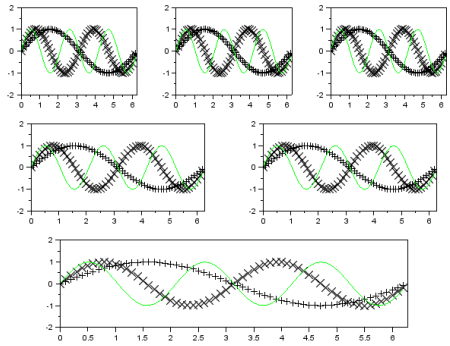
Drugim sposobem jest podział okna graficznego na mniejsze części i umieszenie w nich pojedynczych wykresów. Zadanie to realizowane jest przez polecenie `subplot`. Procedura posiada trzy parametry sterujące, dwa pierwsze definiują liczbę wierszy i kolumn podziału okna, a trzeci numer aktywnego okna. Umiejętnie korzystając z procedury `subplot` można uzyskać różne konfiguracje podziału okna graficznego, jak pokazano w tabeli 3.1.

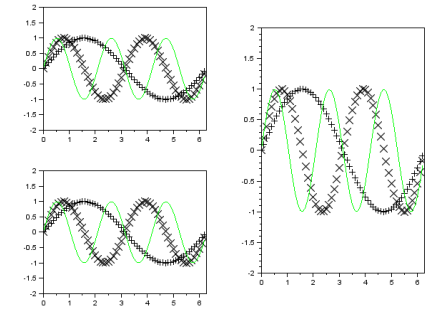
Zauważyć można, że parametry procedury `subplot` można zapisać na dwa sposoby: z lub bez przecinków. Przecinki znajdują zastosowanie, gdy wartość jednego z parametrów przekroczy 9.

Tabela 3.1 Zastosowanie procedury `subplot` do podziału okna graficznego

Kod	Konfiguracja podziału okna graficznego

`subplot(2,3,1);`
`plot2d();`
`subplot(2,3,2);`
`plot2d();`
`subplot(2,3,3);`
`plot2d();`
`subplot(2,3,4);`
`plot2d();`
`subplot(2,3,5);`
`plot2d();`
`subplot(2,3,6);`
`plot2d();`
	

`subplot(3,3,1);`
`plot2d();`
`subplot(3,3,2);`
`plot2d();`
`subplot(3,3,3);`
`plot2d();`
`subplot(3,2,3);`
`plot2d();`
`subplot(3,2,4);`
`plot2d();`
`subplot(3,1,3);`
`plot2d();`
	

`subplot(221);`
`plot2d();`
`subplot(223);`
`plot2d();`
`subplot(122);`
`plot2d();`
	

Źródło: opracowanie własne

Ostatnia metoda polega na otwarciu osobnego okna graficznego dla każdego z generowanych przez skrypt wykresu. Zadanie to realizuje procedura `scf`. Procedurę tą wykorzystuje się do utworzenia okna graficznego i ustawienia okna aktywnego. Rozwiązanie to stosuje się w sytuacjach, gdy zbyt duże nagromadzenie wykresów w jednym oknie graficznym sprawia, że są one

nieczytelne, a przez to ich analiza jest utrudniona. Przeznaczenie jednego okna graficznego, na jeden wykres poprawia jego przejrzystość, zapewnia więcej miejsca na ewentualne opisy lub legendę.

```
wykres_1=scf(0); //utworzenie pierwszego okna graficznego
wykres_2=scf(1); //utworzenie drugiego okna graficznego
wykres_3=scf(2); //utworzenie trzeciego okna graficznego,
ostatnie zdefiniowane okno jest oknem aktywnym

plot2d();
scf(wykres_1); // aktywacja pierwszego okna
plot3d();
scf(wykres_2); // aktywacja drugiego okna
plot2d();
```

3.1. Wykresy

Tworzeniem wykresów w środowisku Scilab realizowane jest głównie przez rodzinę funkcji `plot`, jednakże wyspecjalizowanych funkcji rysujących wykresy 2D i 3D jest znacznie więcej [2]. Wybrane z nich zostaną również omówione.

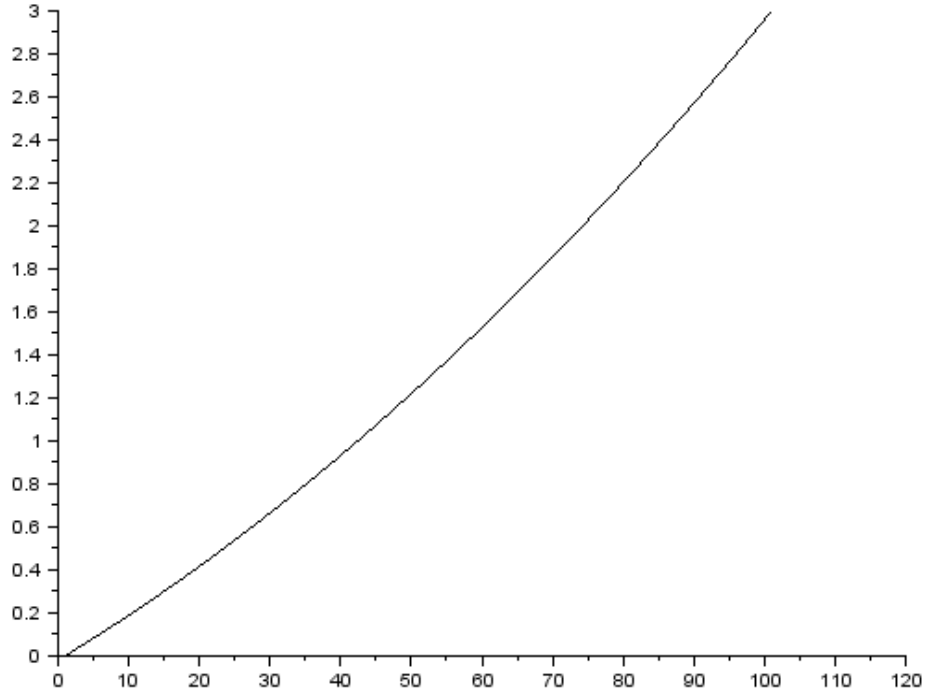
3.1.1. Wykresy 2D

Podstawowym sposobem prezentacji wyników pomiarów lub obliczeń jest wykres dwuwymiarowy przedstawiający przebieg zmian wielkość będącej funkcja zmiennej bazowej w postaci krzywej lub wykres powierzchniowego (mapa, mapa wektorowa) funkcji dwóch zmiennych.

a. Wykresy liniowe

Wykresy liniowe realizowane są za pomocą rodziny procedur `plot2d`. Podstawowe wywołanie procedury wymaga jednego parametry, wektora zawierającego dane reprezentujące wartości odkładane dla osi Oy. Współrzędnymi osi Ox są indeksy kolejnych wartości wektora wejściowego.

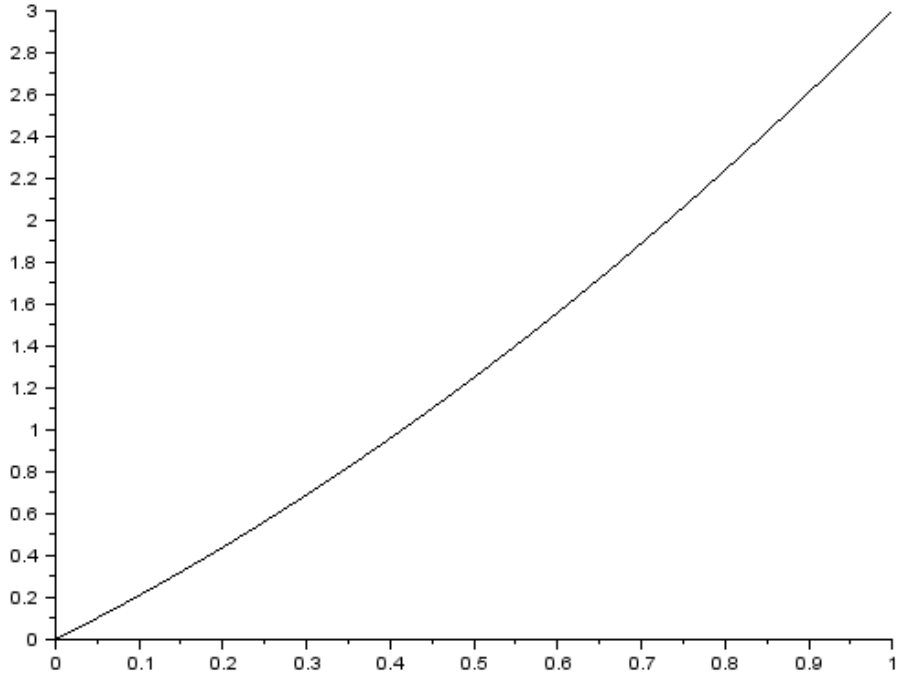
```
clear;clc;clf;
x=0:0.01:1;
y=x^2+2*x;
plot2d(y);
```



Wykres 3.2. Wykres wygenerowany przez skrypt, skala osi x zgodna z liczbą elementów wektora x
Źródło: opracowanie własne

Należy zwrócić uwagę na skalę osi poziomej Ox, wyświetla wartości od 0 do 120, pomimo że przebieg funkcji liczony był od 0 do 1. Dodając do procedury plot drugi parametr wejściowy, wektor zmiennej sterującej, uzyskać można rozkład wartości na osi Ox zgodny z rozkładem danej wejściowej.

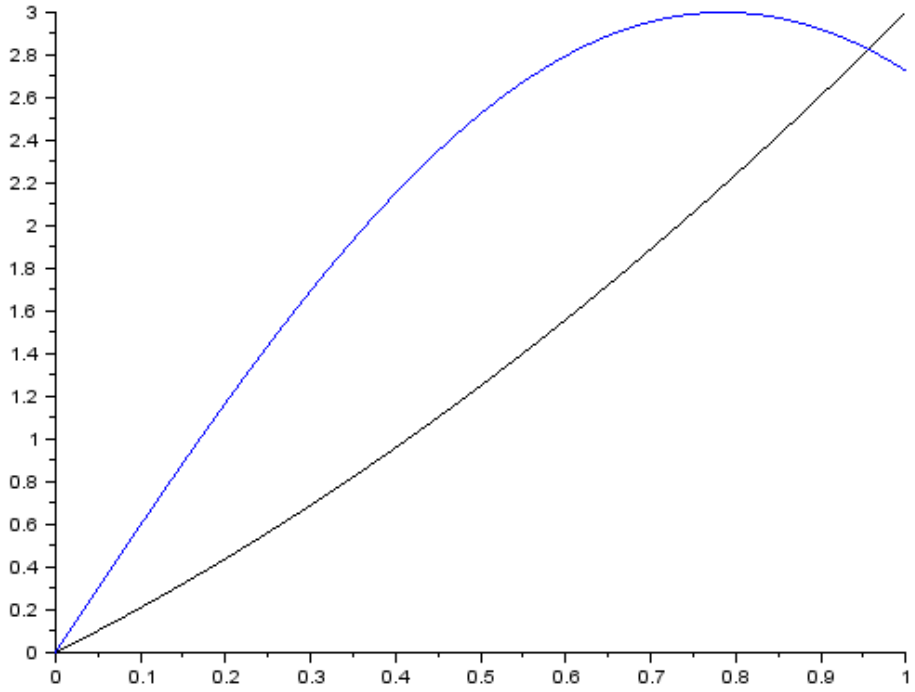
```
clear;clc;clf;
x=0:0.01:1;
y=x^2+2*x;
plot2d(x,y);
```



Wykres 3.3. Wykres wygenerowany przez skrypt, skala osi x zgodna z wartościami wektora x
Źródło: opracowanie własne

Procedury `plot2d` pozwalają na wykreślanie kilku charakterystyk jednej zmiennej na wspólnym wykresie. Należy zwrócić uwagę na transponowanie wektorów `y` i `z`. Procedura ta jest konieczna do prawidłowego wykonania funkcji rysowania wykresów.

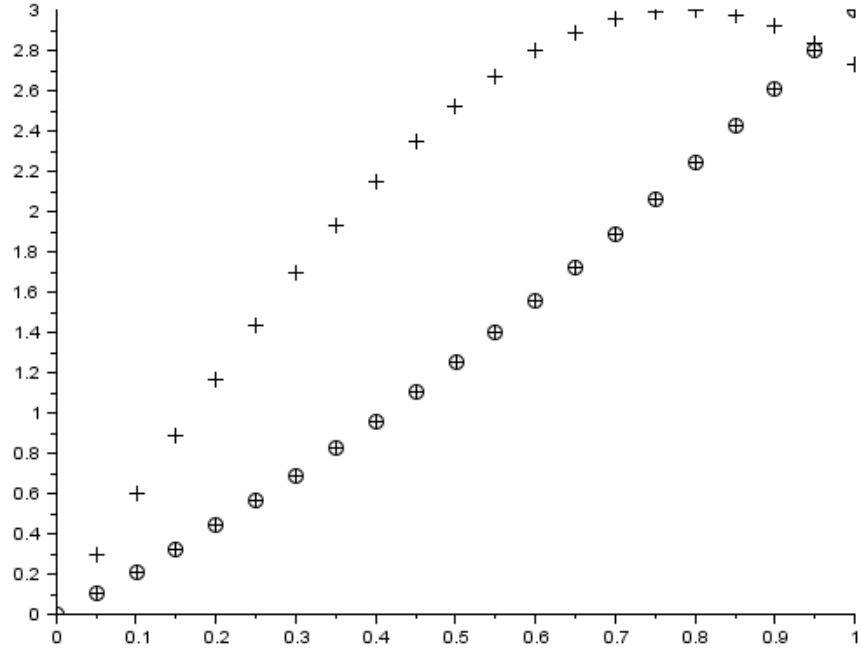
```
clear;clc;clf;
x=0:0.01:1;
z=3*sin(2*x);
y=x^2+2*x;
plot2d(x,[y' z']);
```



Wykres 3.4. Wykres wygenerowany przez skrypt, kilka charakterystyk na wspólnym wykresie
Źródło: opracowanie własne

Procedura `plot2d` automatycznie ustawia różne kolory oraz formaty linii dla kolejnych charakterystyk rysowanych na wykresie. Możliwe jest ręczne zdefiniowanie parametrów wizualnych poszczególnych charakterystyk. Rysowana charakterystyka może być ciągła (liniowa) lub punktowa, wybierana poprzez ustawienie właściwej wartości trzeciego parametru funkcji `plot2d`. W tabeli 3.2 zestawiono wybrane z możliwych do zdefiniowania kolorów linii oraz kształtów rysowanych punktów wraz z odpowiadającymi im wartościami parametrów.

```
clear;clc;clf;
x=0:0.05:1;
z=3*sin(2*x);
y=x^2+2*x;
plot2d(x,[y' z'],[ -3 -15]);
```



Wykres 3.5. Wykres wygenerowany przez skrypt, punkty obliczeniowe wykreślanych przebiegów
Źródło: opracowanie własne

Tabela 3.2. Parametr wizualizacyjny charakterystyki rysowanej procedurą `plot2d`

Kolor	Parametr	Kształt	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

Źródło: opracowanie własne

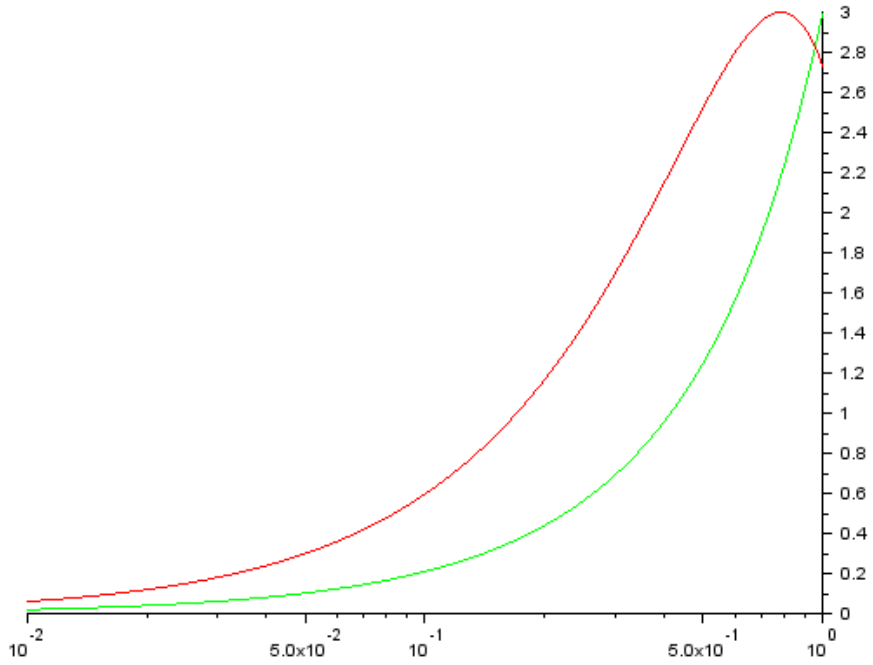
Kolejnym z możliwych do zdefiniowania efektów rysowania wykresów jest definicja skali osi, liniowa lub logarytmiczna, oraz umiejscowienie osi na wykresie. Pierwsze zadanie realizowane jest za pomocą flagi `logflag` definiowanej za pomocą dwóch parametrów, rozkładu liniowego `n` i logarytmicznego `l`. Pierwsza definiowana jest oś Ox. Ustawienie sposobu wyświetlania osi realizuje się za pomocą flagi `axesflag` przyjmującą jeden parametr o wartościach zdefiniowanych w tabeli 3.3.

Tabela 3.3. Parametr wizualizacyjny charakterystyki rysowanej procedurą `plot2d`

Wartość parametru	Opis
0	Nie rysuje osi
1	Oś Oy po lewej stronie, wykres otoczony ramką
2	Wykres otoczony ramką, bez osi
3	Oś Oy po prawej strony
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	Ustawienie domyślne, oś Oy po lewej stronie

Źródło: opracowanie własne

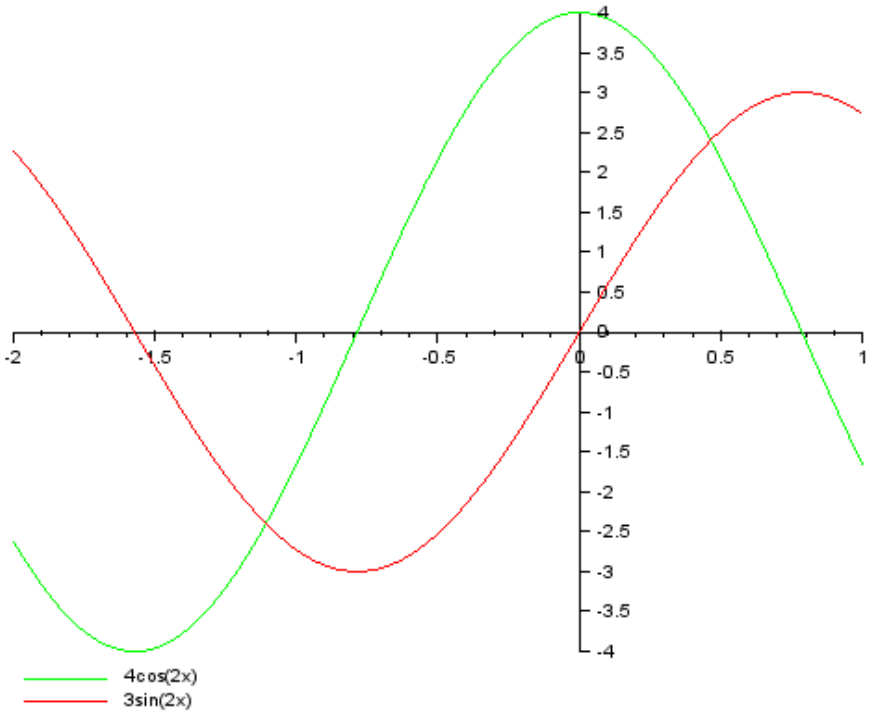
```
clear;clc;clf;
x=0.01:0.001:1;
z=3*sin(2*x);
y=x^2+2*x;
plot2d(x,[y' z'],[ 3 5], logflag="ln", axesflag=3);
```



Wykres 3.6. Wykres wygenerowany przez skrypt
Źródło: opracowanie własne

Niestety za pomocą flagi `axesflag` nie można narysować wykresu w którym osie Ox i Oy przecinają się w punkcie (0,0). Efekt ten można osiągnąć definiując programowo właściwości osi za pomocą funkcji `gca()`. Procedura jest dwuetapowa. W pierwszym etapie należy utworzyć zmienną sterującą, aby w następnym etapie ustawić jej właściwość `location`. Kolejnym istotnym elementem każdego wykresu jest opis rysowanych charakterystyk i osi. Opis rysowanych wykresów zrealizować można poprzez flagę `leg` funkcji `plot2d`. Flaga zawiera tyle nazw, ile wykreślonych jest na wykresie charakterystyk. Poszczególne nazwy należy oddzielać znakiem "@".

```
clear;clc;clf;
x=-2:0.001:1;
z=3*sin(2*x);
y=4*cos(2*x)
plot2d(x,[y' z'],[3 5], leg="4cos(2x)@3sin(2x)");
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
```



Wykres 3.7. Wykres wygenerowany przez skrypt
Źródło: opracowanie własne

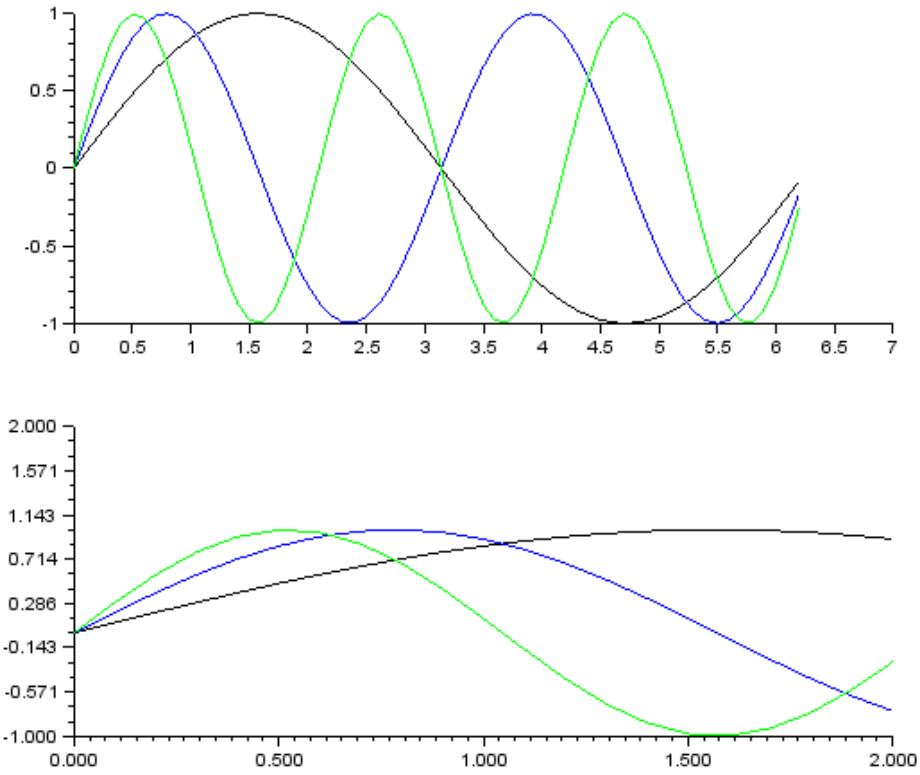
Ostatnimi z omawianych elementów definiujących wykres w funkcji `plot2d` są flagi definiujące zakres wykreślanego obszaru oraz skalowanie osi.

Flaga `rect` definiuje wymiar wykreślanego obszaru. Wymaga ona zdefiniowania czerech parametrów odpowiadających współrzędnym lewego – dolnego i prawego – górnego rogu, prostokąta ograniczającego wykreślany obszar.

Flaga `nax` zaś reguluje podziałkę osi, poprzez zdefiniowanie podziału głównego i podpodziału cząstkowego. Procedura określana jest przez cztery parametry po dwa dla każdej osi. Pierwsza para odnosi się od osi Ox, gdzie pierwszy parametr definiuje liczbę znaczników podpodziału, a drugi liczbę znaczników podziału głównego.

```
clear;clc;clf;
x=[0:0.1:2*%pi]';
subplot(211);
plot2d(x,[sin(x) sin(2*x) sin(3*x)],[1,2,3]);
// wykres wzorcowy
subplot(212);
```

```
plot2d(x,[sin(x) sin(2*x) sin(3*x)], [1,2,3],
  nax=[12,5,2,8], rect=[0,-1,2,2]);
// wykres ograniczony
```



Wykres 3.8. Wykres wygenerowany przez skrypt
Źródło: opracowanie własne

Opis wykresu i osi realizują funkcje `title`, `xlabel`, `ylabel` i `zlabel`. Składnia każdej z nich jest taka sama, składa się z trzech parametrów, przy czym tylko pierwszy, zawierający wyświetlany tekst, jest wymagany, pozostałe dwa to nazwa flagi i jej wartość, za pomocą których określa się właściwości wizualne wyświetlanego przez procedurę tekstu. W tabeli 3.4 zestawiono najistotniejsze flagi poleceń opisu wykresu i osi wraz z formatem zapisu wartości przyjmowanej przez daną flagę.

Tabela 3.4. Flagi i ich zawartość dla funkcji title, xlabel, ylabel i zlabel.

Flaga	Opis	Wartości
`color`	Kolor czcionki	Nazwa koloru: 'red', green', 'blue' itd
`fontsize`	Rozmiar czcionki	Od 0 do 5 co odpowiada: 8pt, 10pt, 12pt, 14pt, 18pt i 24pt.
`fontname`	Nazwa czcionki	Od 0 do 6 co odpowiada: "Courrier", "Symbol", "Times", "Times Italic", "Times Bold", "domyślna"
`position`	Pozycja w oknie graficznym	[x y], gdzie x i y to współrzędne pierwszego znaku opisu, wartości przyjmuje na podstawie wykreślanego wykresu
`rotation`	Obraca opisem	Przyjmuje wartości kąta obrotu w stopniach

Źródło: opracowanie własne

Można zdefiniować właściwości kilku flag dla procedury, należy je podawać sekwencyjne w zapisie wywołującym procedurę, nazwa flagi i następnie wartość przyjmowana przez flagę. Opis wykreślanych na wykresie przebiegów można zamieścić w legendzie za pomocą procedury `legends`. Procedura ta ma trzy wymagane parametry sterujące. Pierwszy zawiera wektor nazw wykreślanych wykresów, elementy wektora oddzielane są znakiem `";"`. Drugi parametr to wektor symboli definiujących formę wyświetlanego opisu wykresu, jego zapis musi być zgodny z przyjętym w procedurze `plot2d`. Ostatni parametr, `opt` definiuje położenie legendy w oknie graficznym, w tabeli 3.5 zestawiono wartości tego parametru wraz z opisem jego właściwości, w postaci tekstowej i liczbowej.

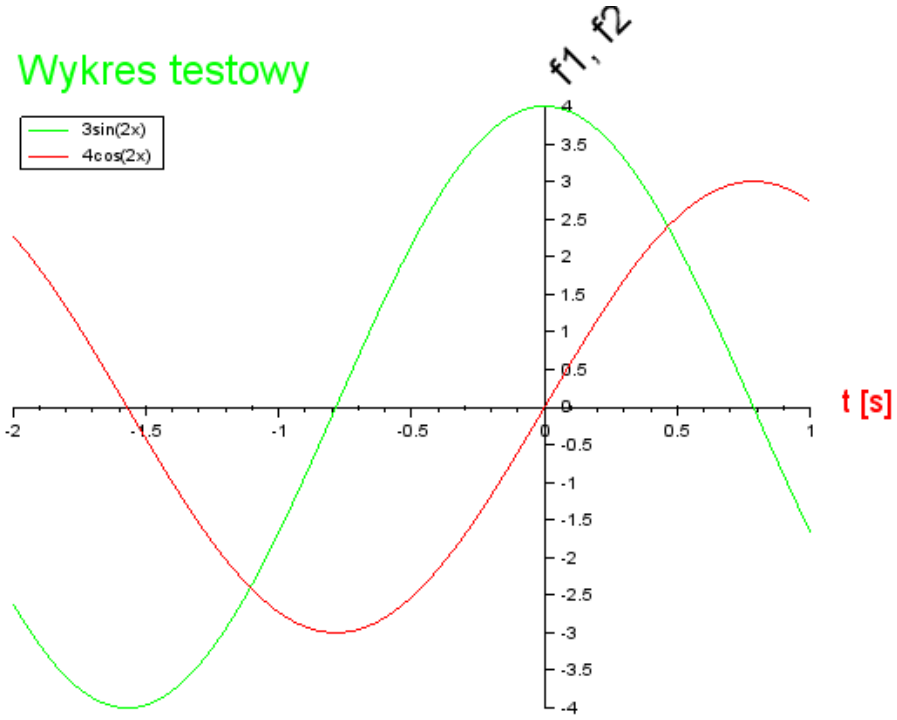
Tabela 3.5. Flagi i ich zawartość dla funkcji `title`, `xlabel`, `ylabel` i `zlabel`.

Opis	Cyfry	Tekst
Prawy górny róg	1	ur
Lewy górny róg	2	ul
Lewy dolny róg	3	ll
Prawy dolny róg	4	lr
Wybór użytkownika	5	?

Źródło: opracowanie własne

```
clear;clc;clf;
x=-2:0.001:1;
z=3*sin(2*x);
y=4*cos(2*x);
plot2d(x,[y' z'],[3 5]);
legends(['3sin(2x)';'4cos(2x)'],[3 5],opt='ul')
```

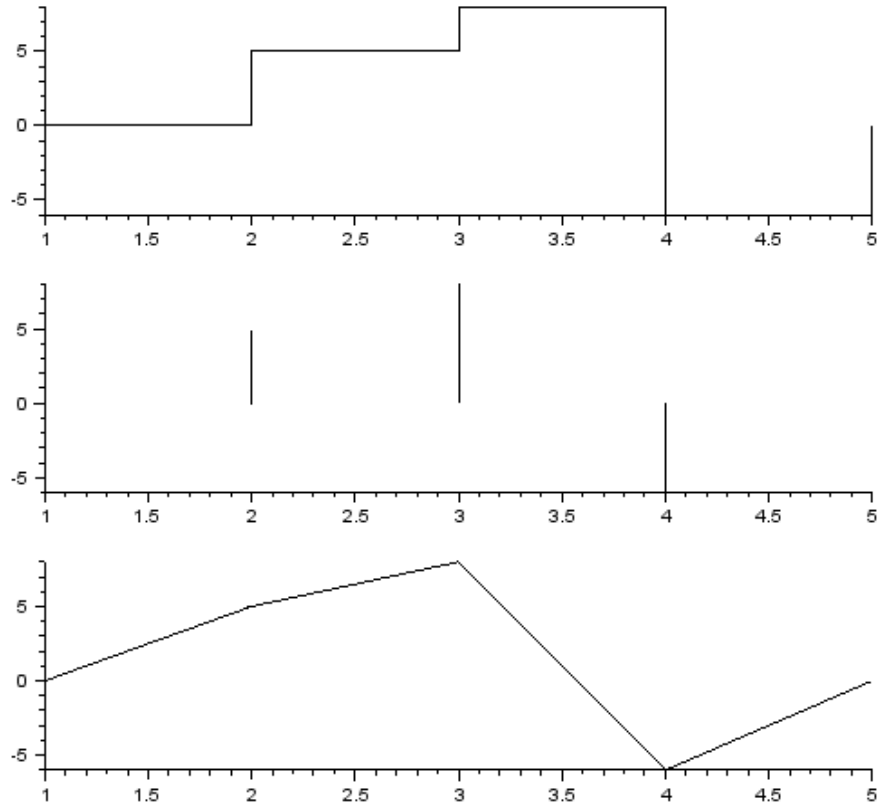
```
title('Wykres testowy', 'fontsize', 5, 'color',
 'green','position', [-2 4.1]);
xlabel('t [s]', 'position', [1.1 -0.2], 'color', 'red',
 'fontsize', 4);
ylabel('f1, f2', 'position', [0.1 4.1], 'rotation', 45,
 'fontname', 0, 'fontsize', 5);
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
```



Wykres 3.9. Wykres wygenerowany przez skrypt
Źródło: opracowanie własne

W systemie Scilab istnieją cztery funkcje pokrewne do `plot2d`, `plot2d1`, `plot2d2`, `plot2d3` i `plot2d4`. Pierwsza z funkcji jest konstrukcją przestarzałą, praktycznie już niestosowaną, funkcjonalnością odpowiadającą podstawowej funkcji `plot2d`. Pozostałe trzy funkcje wykreślają wektory danych wejściowych w sposób indywidualny. Opis parametrów procedury jest tożsamy z opisem procedury `plot2d`. Funkcja `plot2d2` wykreśla wykres schodkowy – przyrostowy, `plot2d3` charakterystykę impulsową, zaś `plot2d4` każdy z wykreślanych przyrostów jest w postaci wektora.

```
clear;clc;clf;
x=[0,5,8,-6,0];
subplot(311); plot2d2(x);
subplot(312); plot2d3(x);
subplot(313); plot2d4(x);
```



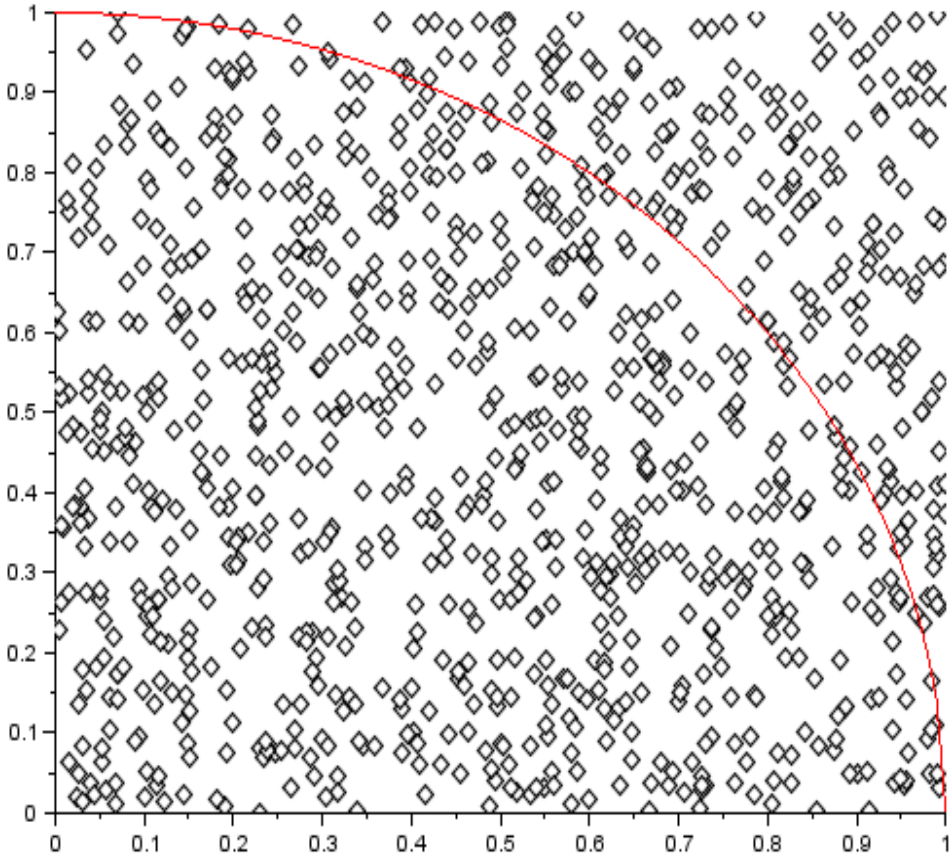
Wykres 3.10. Wykres wygenerowany przez skrypt
Źródło: opracowanie własne

b. Wykresy powierzchniowe

Wykresy powierzchniowe są graficzną formą dwuwymiarowej prezentacji wielkości będącej funkcją dwóch zmiennych. Wielkość wykreślanej wartości prezentowana jest w formie zero-jedynkowej lub mapy w postaci barwnej lub konturowej.

Pierwszy typ wykresu powierzchniowego można wykreślić za pomocą funkcji `plot2d` ustawiając styl rysowania na punktowy. Zaprezentowany przykład będzie wyznaczać przybliżoną wartość liczby Pi.

```
clear; clc; clf;
format('v',7)
t=0;
N=1000;
rand("uniform")
for i=1:N
    x=rand(1,1);
    y=rand(1,1);
    xi(1,i)=x;
    yi(1,i)=y;
    zi(1,i)=sqrt(x^2+y^2)
    r1=(x^2+y^2)
    if r1<=1 then
        t=t+1
    end
end
pi=(4*t)/N;
disp(pi)
plot2d(xi,yi,style=-5);
z=0:0.01:1;
plot2d(z,sqrt(1-z^2),5)
```



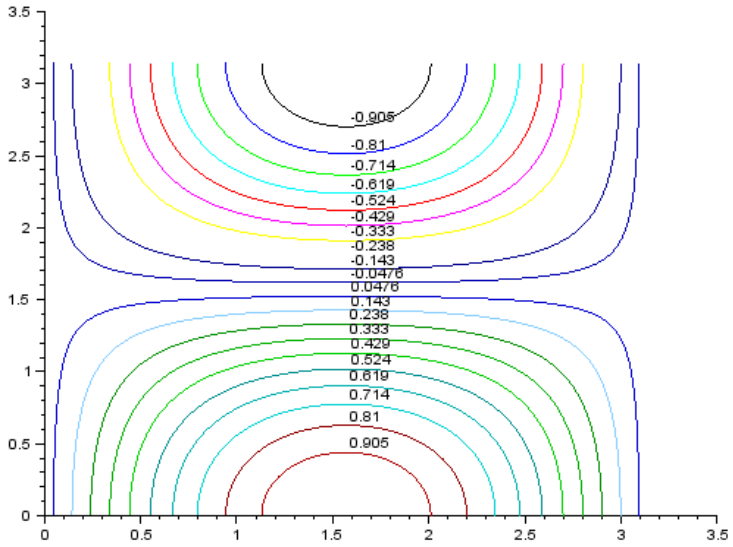
Wykres 3.11. Wykres powierzchniowy (0,1)
Źródło: opracowanie własne

Drugi typ wykresów mapowych realizuje się za pomocą wyspecjalizowanych funkcji, spośród których omówione zostaną `contourf` i `sfgrayplot` oraz wspomagająca funkcja generująca legendę `colorbar`.

W obydwu funkcjach wartość przyjmowana w każdym punkcie wykresu przekazywana jest za pomocą zdefiniowanej w skrypcie funkcji lub macierzy.

Funkcja `contourf` wymaga czterech podstawowych parametrów. Można zdefiniować także parametry opcjonalne ustawiające niektóre parametry wizualnej formy prezentowanej mapy. Pierwsze dwa parametry definiują wartości liczbowe punktów dla jakich wykreślany jest wykres. Trzeci parametr zawiera wskazanie na funkcje określającą wartości przyjmowane w poszczególnych punktach wykreślanego wykresu. Czwarty parametr określa ilość konturów użytych w rysowanym wykresie.

```
clear;clc;clf;
function z=surf(x, y)
    z=cos(y)*sin(x)
endfunction
x=linspace(0,%pi,500)
// definicja wektora danych osi Ox
y=linspace(0,%pi,200)
// definicja wektora danych osi Oy
contour(x,y,surf,20)
// wykreślanie wykresu konturowego
```



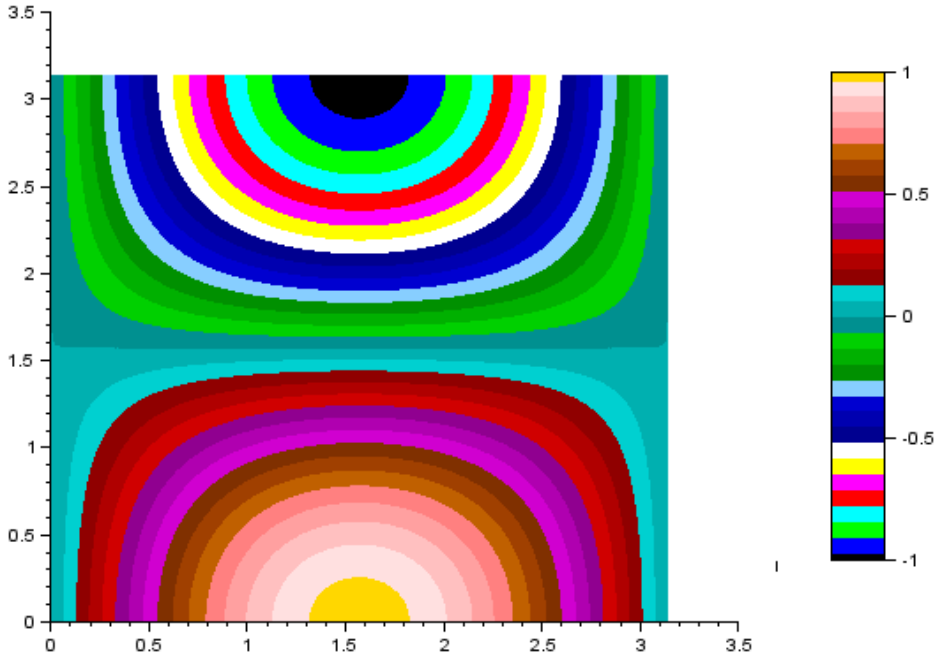
Wykres 3.12. Wykres powierzchniowy – konturowy
Źródło: opracowanie własne

Funkcja `sfgrayplot` pozwala na wygenerowanie wykresu powierzchniowego typu mapowego. Wartość wykreślanej funkcji lub macierzy w punkcie prezentowana jest za pomocą odzienia barwy. Skalę barwną wykresu umieszcza się za pomocą funkcji `colorbar`.

Funkcja `sfgrayplot` wymaga w podstawowej wersji wywołania trzech parametrów, dwa pierwsze to wektory określające wartości przyjmowane przez współrzędne x i y generowanego wykresu. Trzecim jest wskazanie funkcji określającej wartości przyjmowanej w punktach określanych przez wektory x i y. Pozostałe z dostępnych parametrów są opcjonalne, szczegóły na ich temat dostępne w pomocy systemowej, a służą do personalizacji wizualnej formy prezentowanego wykresu.

Umieszczenie w oknie graficznym razem z wykresem barwnej skali w nim zastosowanej wymaga podania dwóch parametrów funkcji `colorbar`. Określają one przedział wartości w jakim wygenerowana zostanie skala, a więc wartość maksymalną i minimalną otrzymaną z funkcji zastosowanej przy wykreślaniu wykresu.

```
clear;clc;clf;
function z=surf(x, y)
    z=cos(y)*sin(x)
endfunction
k=200;
x=linspace(0,%pi,k)
y=linspace(0,%pi,k)
for i=1:k// obliczenie wartości funkcji wymagane do
    for j=1:k // odszukania wartości min i max
        z(i,j)=surf(x(i),y(j))
    end
end
sfgrayplot(x,y,surf); // wykreślanie mapy
colorbar(min(z),max(z)) //wykreślanie skali barwnej
```

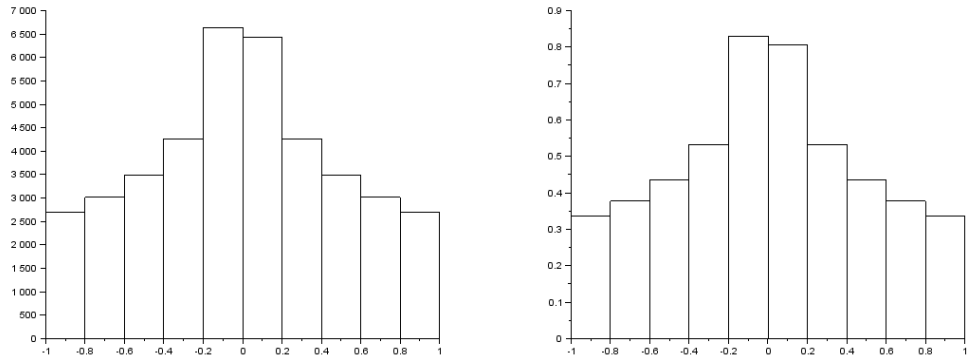


Wykres 3.13. Wykres powierzchniowy – mapa
Źródło: opracowanie własne

c. Histogram

W środowisku Scilab zdefiniowano szereg wyspecjalizowanych funkcji służących do generowania wykresów. Na szczególną uwagę zasługuje funkcja `histplot`, która analizuje wejściowy wektor danych i następnie wykreśla histogram prezentujący statystkę jego zawartości. Funkcja jest wywoływana z dwoma wymaganymi parametrami, pierwszy określa liczbę przedziałów analizy, a drugim jest wektor danych.

```
clear;clc;clf;
function z=surf(x, y)
    z=cos(y)*sin(x)
endfunction
k=200;
x=linspace(0,%pi,k)
y=linspace(0,%pi,k)
for i=1:k
    for j=1:k
        z(i,j)=surf(x(i),y(j))
    end
end
subplot(1,2,1)
histplot(10,z,normalization=%f)
subplot(1,2,2)
histplot(10,z)
```



Wykres 3.14. Wykresy typu histogram znormalizowany i nie znormalizowany
Źródło: opracowanie własne

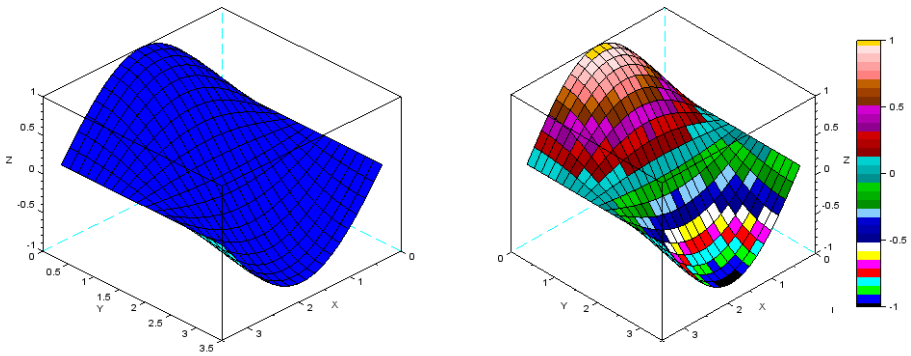
Trzeci parametr umożliwia wygenerowanie histogramu znormalizowanego.

3.1.2. Wykresy 3D

Wykresy 3D znajdują podobne zastosowanie jak wykresy mapowe 2D. Ich wyższość polega na tym, że poza informacją barwną o przyjmowanej wartości, dodatkowo wychylenie w osi z informuje o przyjmowanej w danym punkcie wartości.

Kreślenie wykresów 3D realizowane jest za pomocą funkcji `plot3d`. Wymaga ona przy wywołaniu podania trzech parametrów, dwa pierwsze to wektory definiujące osie Ox i Oy, trzecim parametrem jest macierz lub wskazanie na funkcję określającą wartości wychylenia wykresu w osi Oz. Funkcja ta, wartość w danym punkcie obrazuje poprzez wychylenie, chcąc pokazać też wartość w danym punkcie za pomocą skali barwnej należy zastosować funkcje `plot3d1`.

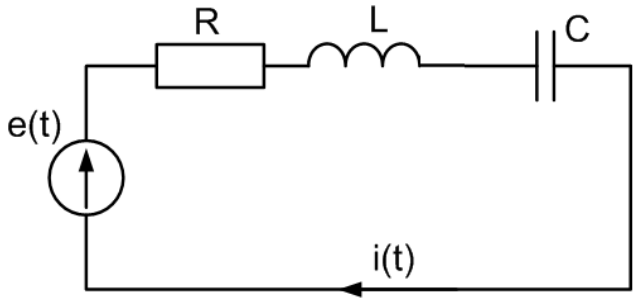
```
clear;clc;clf;
function z=surf(x, y)
    z=cos(y)*sin(x)
endfunction
k=20;
x=linspace(0,%pi,k)
y=linspace(0,%pi,k)
for i=1:k
    for j=1:k
        z(i,j)=surf(x(i),y(j))
    end
end
subplot(121)
plot3d(x,y,z)
subplot(122)
colorbar(min(z),max(z))
plot3d1(x,y,z)
```



Wykres 3.15. Wykresy 3D, uzyskane za pomocą poleceń `plot3d` i `plot3d1`
Źródło: opracowanie własne

Zadania

Praktyczne zastosowanie wiadomości omówionych w niniejszym rozdziale zobrazować można na przykładzie analizy prostego obwodu RLC prądu przemiennego, jak pokazano na rysunku 3.1 [4].



Rys. 3.1. Schemat przykładowego obwodu RLC
Źródło: opracowanie własne

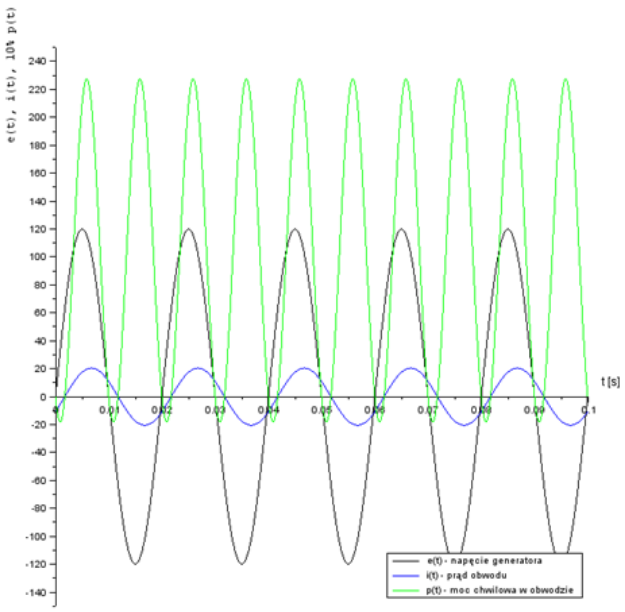
Dla pokazanego powyżej obwodu można napisać skrypt realizujący podstawowe zagadnienia przewidziane przy analizie takiego obwodu:

a) Wartości chwilowe prądu i mocy
b) Rysowanie przebiegów prądu, napięcia i mocy
c) Wartości zespolone i skuteczne prądu i napięć
d) Wykres fazorowy

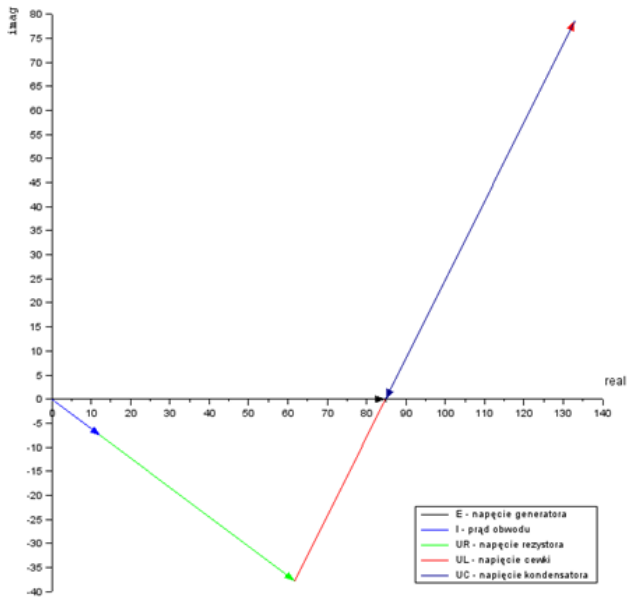
```
clear;clc;clf;
Em=120;
R=5;
L=30*10^-3;
C=0.5*10^-3;
f=50;
omega=2*%pi*f;
function e=napiecie(t)
    e=Em*sin(omega*t);
endfunction
function i=prad(t)
    i=Im*sin(omega*t+fiI);
endfunction
function p=moc(t)
    p=napiecie(t).*prad(t);
endfunction
t=linspace(0,5/f,1000);
E=Em/sqrt(2);
```

```
  Z=R+%i*(omega *L-1/(omega *C));
  I=E/Z;
  Im=abs(I)*sqrt(2);
  fiI=atan(imag(I),real(I));
  UR=I*R;
  UL=I*omega*L*%i;
  UC=I/(%i*omega*C);
  subplot(121);
  plot2d(t,[napiecie(t)' prad(t)' 0.1*moc(t)'])
  ster_axe=gca();
  ster_axe.x_location="origin";
  ster_axe.y_location="origin";
  legends(['e(t)  -  napęcie  generatora';'i(t)  -  prąd
obwodu';'p(t) - moc chwilowa w obwodzie'],[1 2 3],opt='lr');
  xlabel('t [s]', 'position', [0.102 5],'fontsize', 2);
  ylabel('e(t),  i(t),  10%  p(t)',  'position',  [-0.007
180],'fontname', 0, 'fontsize', 2,'rotation', 90);
  subplot(122);
  URx=[0,real(UR)];
  URy=[0,imag(UR)];
  ULx=[real(UR),real(UL)+real(UR)];
  ULy=[imag(UR),imag(UL)+imag(UR)];
  UCx=[real(UL)+real(UR),real(UC)+real(UL)+real(UR)];
  UCy=[imag(UL)+imag(UR),imag(UC)+imag(UL)+imag(UR)];
  plot2d4(URx,URy,3);
  plot2d4(ULx,ULy,5);
  plot2d4(UCx,UCy,9);
  Ex=[0,E];
  Ey=[0,0];
  plot2d4(Ex,Ey);
  Ix=[0,real(I)];
  Iy=[0,imag(I)];
  plot2d4(Ix,Iy,2);
  legends(['E - napęcie generatora';'I - prąd obwodu';'UR -
napęcie  rezystora';'UL  -  napięcie  cewki';'UC  -  napięcie
kondensatora'],[1 2 3 5 9],opt='lr');
  ster_axe=gca();
  ster_axe.x_location="origin";
  ster_axe.y_location="origin";
  xlabel('real', 'position', [140 2.5], 'fontsize', 2);
  ylabel('imag',  'position',  [-8  75],  'rotation',  90,
'fontname', 0, 'fontsize', 2);
```

Skrypt wykreśla dwa wykresy, przebiegów chwilowych prądu, napięcia i mocy oraz wykres wskazowy prądu i napięć w obwodzie, jak pokazano na wykresach 3.16 i 3.17.



Wykres 3.16. Przebiegi prądu, napięcia i mocy wykreślone dla analizowanego obwodu
Źródło: opracowanie własne



Wykres 3.17. Wykresy fazorowy wykreślony przez skrypt
Źródło: opracowanie własne

1. W oparciu o przykładowy skrypt napisać program analizujący rozgałęziony obwód RLC zawierający kilka elementów R, L i C zasilany ze źródła napięcia przemiennego.
2. Wyznaczyć rozkład natężenia pola elektrycznego [4] w otoczeniu układu kilku ładunków punktowych umieszczonych na płaszczyźnie. Natężenie pola elektrycznego obliczyć można z zależności $E = \frac{|Q|}{4\pi\varepsilon r^2}$, jednakże przy wyznaczaniu wartości natężenia pola w punkcie od kilku ładunków należy pamiętać, że sumowanie musi być wykonane wektorowo. Bezpośrednie otoczenie ładunku musi zostać pominięte w obliczeniach ($\frac{1}{r^2} \to \infty$ przy $r \to 0$).
3. Napisać skrypt wykreślający rozkład natężenia pola magnetycznego wytworzonego przez zdefiniowaną przez siebie ramkę z prądem I. Skrypt oblicza rozkład H w otoczeniu ramki z prądem na powierzchni na której znajduje się ramka. W zadaniu należy wykorzystać prawo Biota-Savarta [4], gdzie wartość natężenia pola w punkcie wyznacza się ze wzoru $H = \frac{I}{2\pi r}(\cos(\alpha) - \cos(\beta))$. Efektem działania skryptu będą mapy rozkładu H oraz wykres 3D obrazujący rozkład H.
4. Należy pamiętać że model nie daje możliwości obliczenie natężenia pola w przewodzie, należy przyjąć że średnia przewodu ramki jest nieskończenie mała, oraz pominąć w obliczeniach obszar w pobliżu przewodów.

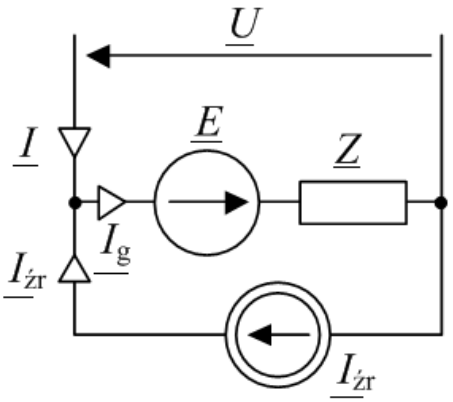
4. Macierzowa analiza obwodów elektrycznych

Gałąź obwodu elektrycznego tworzy jeden lub kilka połączonych ze sobą szeregowo elementów idealnych. Cechą charakterystyczną gałęzi jest więc natężenie prądu.

Każda gałąź włączona jest pomiędzy dwa węzły obwodu. Cechą szczególną węzła jest jego potencjał elektryczny. Węzeł obwodu elektrycznego to końcówka (zacisk) obwodu, do której przyłączone są dwie lub więcej gałęzi.

Oczko obwodu elektrycznego to połączenia gałęzi tworzące kontur zamknięty, we wnętrzu którego nie może znajdować się już żadna inna gałąź.

Powyższe pojęcia wymagają jednak rozszerzenia. Tzw. gałąź uogólniona oprócz elementów pasywnych może zawierać równocześnie elementy źródłowe obu typów: prądowy i napięciowy. Gałąź uogólniona obwodu elektrycznego została przedstawiona na rysunku 4.1 [8].



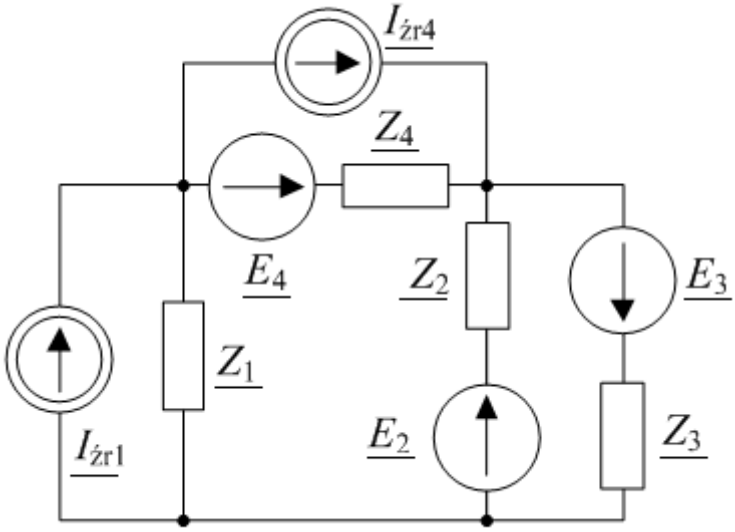
Rysunek 4.1. Gałąź uogólniona obwodu elektrycznego
Źródło: opracowanie własne na podstawie [8].

Na rysunku 4.2 pokazano przykładowy schemat obwodu prądu sinusoidalnie przemiennego. Pamiętając o pojęciu gałęzi uogólnionej można zauważyć, że ma on cztery gałęzie i trzy węzły.

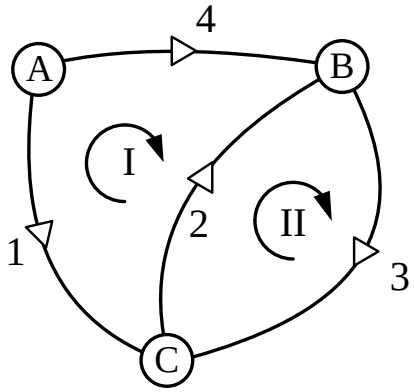
Dla każdego obwodu elektrycznego można narysować uproszczoną strukturę geometryczną składającą się z punktów oznaczających węzły i odcinków symbolizujących gałęzie. Struktura taka, zawierająca numerację składników, nazywana jest grafem niezorientowanym obwodu. Jeżeli zostanie uzupełniona o oznaczenia kierunków prądów w gałęziach oraz kierunków obiegowych oczek to powstanie graf zorientowany obwodu (rysunek 4.3).

Wybór kierunków prądów w gałęziach i wybór kierunków obiegowych oczek jest dowolny. Warto jednak pamiętać, że w prostych konfiguracjach kierunek prądu może być przewidywany na podstawie kierunków działania wymuszeń

napięciowych i prądowych. Natomiast obiegi oczek najwygodniej jest oznaczać jednakowo np. jako zgodne z kierunkiem ruchu wskazówek zegara.



Rysunek 4.2. Schemat przykładowego obwodu elektrycznego
Źródło: opracowanie własne



Rysunek 4.3. Graf zorientowany dla obwodu elektrycznego z rysunku 4.2
Źródło: opracowanie własne

4.1. Macierz strukturalna węzłowa

Strukturę zorientowanego grafu obwodu (rysunek 4.3) można przedstawić w postaci macierzy. Wierszom macierzy odpowiadają węzły, kolumnom macierzy – gałęzie. Elementami macierzy strukturalnej mogą być wartości -1, 0 i 1. Jeżeli gałąź nie łączy się z węzłem należy przyjąć wartość 0. Może to zaskoczyć studentów teorii obwodów ale jeżeli gałąź jest związana z węzłem i jej

zwrot jest skierowany od tego węzła (prąd wypływa z węzła) należy przyjąć wartość 1. Jeśli zwrot jest do węzła – wartość -1.

Pełna macierz incydencyjna węzłowa $\mathbf{A}_p$ dla grafu z rysunku 2.3 wygląda następująco:

$$\mathbf{A}_p = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & -1 \\ -1 & 1 & -1 & 0 \end{bmatrix}.$$

Charakterystyczne jest to, że w powyższej macierzy suma wartości wyznaczona dla każdej kolumny wynosi 0.

Wybierając węzeł odniesienia (uziemiony) można utworzyć macierz incydencyjną węzłową **A**. Przykładowo, przy założeniu, że węzeł C z rysunku 4.3 został uziemiony wygląda ona następująco:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 1 & -1 \end{bmatrix}.$$

4.2. Macierz strukturalna oczkowa

Macierz incydencyjną oczkową wyznacza się analizując związek gałęzi z oczkami. Wierszom macierzy odpowiadają oczka, kolumnom – ponownie gałęzie. Jeżeli gałąź nie należy do danego oczka należy przyjąć 0. Jeśli kierunek gałęzi tworzącej oczko pokrywa się z obiegiem danego oczka wartość opisująca ten związek wynosi 1, jeśli kierunki te są przeciwne to wartość wynosi -1.

Dla grafu z rysunku 4.3 macierz incydencyjna oczkowa **B** wygląda następująco:

$$\mathbf{B} = \begin{bmatrix} -1 & -1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Dobrym sposobem sprawdzenia poprawności opisu grafu jest wykorzystanie faktu, że powyższe macierze strukturalne spełniają następujące równania:

$$\mathbf{A}\mathbf{B}^T = 0$$

$$\mathbf{B}\mathbf{A}^T = 0$$

gdzie wskaźnik *T* oznacza transpozycję macierzy.

4.3. Macierz diagonalna impedancji (admitancji)

Macierz diagonalna impedancji opisuje przynależność elementów pasywnych do gałęzi obwodu elektrycznego Dla obwodu z rysunku 4.2 i jego grafu zorientowanego pokazanego na rysunku 2.3 macierz diagonalna impedancji $\mathbf{Z}_d$ ma następującą postać

$$\mathbf{Z}_d = \begin{bmatrix} \underline{Z}_1 & 0 & 0 & 0 \\ 0 & \underline{Z}_2 & 0 & 0 \\ 0 & 0 & \underline{Z}_3 & 0 \\ 0 & 0 & 0 & \underline{Z}_4 \end{bmatrix}.$$

Jej odwrotność to macierz diagonalna admitancji $\mathbf{Y}_d$:

$$\mathbf{Y}_d = \frac{1}{\mathbf{Z}_d} = \begin{bmatrix} \dfrac{1}{\underline{Z}_1} & 0 & 0 & 0 \\ 0 & \dfrac{1}{\underline{Z}_2} & 0 & 0 \\ 0 & 0 & \dfrac{1}{\underline{Z}_3} & 0 \\ 0 & 0 & 0 & \dfrac{1}{\underline{Z}_4} \end{bmatrix}.$$

4.4. Macierze kolumnowe wymuszeń

Macierz kolumnowa (wektor) wymuszeń napięciowych opisuje położenie idealnych źródeł napięcia w gałęziach obwodu. Jeżeli kierunek działania źródła napięciowego jest przeciwny względem zwrotu rozpatrywanej gałęzi to jego wartość trafia do macierzy kolumnowej ze znakiem minus. W przypadku zgodności kierunków do macierzy wpisywana jest wartość **s. em**. źródła. Jeżeli zastosuje się podany wcześniej sposób oznaczania zwrotów gałęzi to ta druga możliwość wystąpi w 100% przypadków.

Dla schematu z rysunku 4.2 i grafu z rysunku 4.3 macierz kolumnowa wymuszeń napięciowych $\mathbf{E}_g$ przyjmuje postać

$$\mathbf{E}_g = \begin{bmatrix} 0 \\ \underline{E}_2 \\ \underline{E}_3 \\ \underline{E}_4 \end{bmatrix}.$$

Idealne źródła prądu będące składnikami gałęzi uogólnionych znajdują swój opis w macierzy kolumnowej wymuszeń prądowych $\mathbf{I}_z$. Należy zwrócić szczególną uwagę na formułowanie zawartości tej macierzy ponieważ jest to przyczyna największej liczby błędów popełnianych przy rozwiązywaniu zadań metoda macierzową. Jeżeli kierunek działania źródła prądu pokrywa się ze zwrotem danej gałęzi to wartość wydajności prądowej brana jest ze znakiem przeciwnym. Jeżeli zwroty są przeciwne to prąd źródłowy brany jest bez zmiany znaku.

Dla schematu z rysunku 4.2 i grafu z rysunku 4.3 macierz kolumnowa wymuszeń prądowych $\mathbf{I}_z$ przyjmuje postać

$$\mathbf{I}_z = \begin{bmatrix} \underline{I}_{\text{źr1}} \\ 0 \\ 0 \\ -\underline{I}_{\text{źr4}} \end{bmatrix}.$$

4.5. Metoda oczkowa

Macierz impedancji własnych i wzajemnych z może być obliczona jako

$$\mathbf{Z} = \mathbf{B} \cdot \mathbf{Z}_d \cdot \mathbf{B}^T .$$

Prądy oczkowe $\mathbf{I}_o$ można obliczyć pod warunkiem, że możliwe jest wyznaczenie macierzy odwrotnej $\mathbf{Z}^{-1}$:

$$\mathbf{I}_o = \mathbf{Z}^{-1} \cdot \mathbf{B} \cdot \left(\mathbf{E}_g - \mathbf{Z}_d \cdot \mathbf{I}_z \right) .$$

Prądy wypadkowe gałęzi uogólnionych $\mathbf{I}$ (na przykład prąd $\underline{I}$ na rysunku 4.1) oblicza się jako

$$\mathbf{I} = \mathbf{B}^T \cdot \mathbf{I}_o$$

Ostatecznie, prądy płynące przez impedancje gałęzi można wyznaczyć z pierwszego prawa Kirchhoffa:

$$\mathbf{I}_g = \mathbf{I} + \mathbf{I}_z$$

4.6. Metoda węzłowa

Macierz admitancji własnych i wzajemnych **Y** oblicza się jako

$$\mathbf{Y} = \mathbf{A} \cdot \mathbf{Y}_d \cdot \mathbf{A}^T \ .$$

Wyznaczenie macierzy potencjałów węzłowych $\mathbf{U}_w$ jest osiągalne pod warunkiem, że można obliczyć macierz odwrotną $\mathbf{Y}^{-1}$:

$$\mathbf{U}_w = \mathbf{Y}^{-1} \cdot \mathbf{A} \cdot \left(\mathbf{I}_z - \mathbf{Y}_d \cdot \mathbf{E}_g \right) .$$

Napięcia na gałęziach obwodu **U** wyznacza się jako

$$\mathbf{U} = \mathbf{A}^T \cdot \mathbf{U}_w \ .$$

Ostatecznie, prądy płynące przez admitancje gałęzi $\mathbf{I}_g$ można wyznaczyć jako

$$\mathbf{I}_\mathrm{g} = \mathbf{I} + \mathbf{I}_z = \mathbf{Y}_d \cdot \left(\mathbf{U} + \mathbf{E}_g \right) .$$

Przykład
Dane liczbowe do schematu z rysunku 2.2:
$\underline{Z}_1$ = 10 Ω, $\underline{Z}_2$ = (10+10i) Ω, $\underline{Z}_3$ = (5-10i) Ω, $\underline{Z}_4$ = -10i Ω,
$\underline{I}_{źr1}$ = (2+i) A, $\underline{I}_{źr4}$ = 1 A, $\underline{E}_2$ = 50 V, $\underline{E}_3$ = (10-20i) V, $\underline{E}_4$ = -50i V

Skrypt rozwiązujący zadanie może wyglądać następująco:

```
// czyszczenie pamięci
clear
// czyszczenie konsoli
clc
// dane: impedancje gałęzi
Z1 = 10;
Z2 = 10+10*%i;
Z3 = 5-10*%i;
```

```
Z4 = -10*%i;

// dane: parametry źródeł
Izr1 = 2+%i;
Izr4 = 1;
E2 = 50;
E3 = 10-20*%i;
E4 = -50*%i;

// macierz incydencyjna węzłowa
A=[1, 0, 0, 1; 0, -1, 1, -1];

// macierz diagonalna admitancji
Yd=diag([1/Z1, 1/Z2, 1/Z3, 1/Z4]);

// macierz kolumnowa źródeł napięcia
Eg=[0; E2; E3; E4];

// macierz kolumnowa źródeł prądu
Iz=[Izr1; 0; 0; -Izr4];

// macierz admitancji własnych i wzajemnych
Y=A*Yd*A';

// potencjały węzłów
Uw=inv(Y)*A*(Iz-Yd*Eg);

// napięcia na gałęziach
U=A'*Uw;

// prądy w gałęziach
Ig=Yd*(U+Eg);

// wartości skuteczne prądów w gałęziach
Igs=abs(Ig);

// wyświetlanie wyników
disp(Ig, 'wartości zespolone prądów');
disp(Igs, 'wartości skuteczne prądów')
```

Zadanie

Wyznaczyć prądy gałęziowe metodą oczkową i węzłową dla obwodu elektrycznego zadanego przez prowadzącego zajęcia.

Sprawozdanie powinno zawierać:

1. stronę tytułową,

2. pełny opis zadania (schemat elektryczny z oznaczeniami parametrów przyznanych grupie przez prowadzącego, graf zorientowany),
3. pełne skrypty Scilab-a (polecenia) rozwiązujące zadanie z komentarzami programisty,
4. wyniki obliczeń (prądy gałęziowe w postaci zespolonej i ich wartości skuteczne),
5. porównanie wyników uzyskanych dwiema metodami,
6. wnioski.

Uwagi:

- Do wydruku sprawozdania należy dołączyć nośnik zawierający wersję elektroniczną plików (skrypty obliczeniowe).

5. Całkowanie i różniczkowanie numeryczne

Całkowanie i różniczkowanie są jednymi z podstawowych i najważniejszych przekształceń matematycznych stosowanych w obliczeniach inżynierskich. Opracowano kilka metod numerycznych realizujących te zadania. W środowiskach obliczeniowych, takich jak *Matlab* i *Scilab* [3], można samodzielnie napisać funkcję realizującej całkowanie lub różniczkowanie numeryczne, lub skorzystać z wbudowanych funkcji realizujących te zadania.

5.1. Całkowanie numeryczne

Całkowanie numeryczne polega na przybliżonym wyznaczaniu całek oznaczonych za pomocą odpowiedniej sumy ważonej wartości całkowanej funkcji w kilku punktach. Podział przedziału całkowania na jak najmniejsze fragmenty znacząco podnosi dokładność uzyskanego wyniku obliczeń. Wynik końcowy jest sumą oszacowań całki w poszczególnych przedziałach. Zazwyczaj podział przedziału całkowania jest równomierny, ale w celu poprawy dokładności obliczeń zagęszcza się podział przedziału całkowania w przedziale o dużej szybkości zmian funkcji. Opracowano kilka metod wyznaczania całek oznaczonych [7].

5.1.1. Metoda prostokątów

Metoda prostokątów opiera się na definicji całki oznaczonej Riemanna, w której wartość całki interpretuje się jako sumę pól obszarów pod wykresem funkcji podcałkowej w zadanym przedziale całkowania <a,b>. Suma ta jest przybliżana za pomocą sumy pól odpowiednio dobranych prostokątów [4]. Przedział całkowania <a,b> dzielony jest na n jednakowych przedziałów.

Długość jednego kroku podziału przedziału całkowania wyznaczana jest na podstawie zależności:

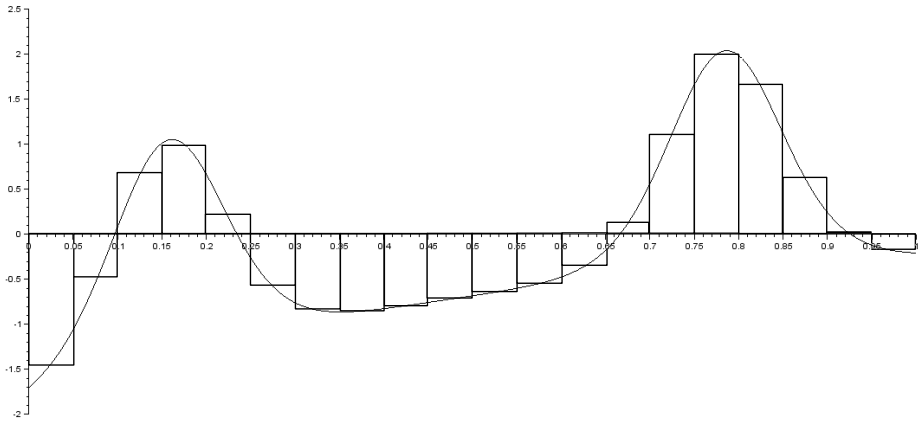
$$dx = \frac{b-a}{n} \tag{1}$$

W każdym wyznaczonym punkcie podziału wyznaczana jest wartość funkcji podcałkowej. Wartość całki oznaczonej w efekcie wyznacza się z zależności:

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=1}^{n} f\left(a + i\cdot\frac{b-a}{n}\right) \tag{2}$$

Graficzną prezentacje działania metody prostokątów obrazuje wykres 5.1, wykreślony dla funkcji zapisanej równaniem (3) w przedziale (0,1):

$$f(t) = 0.2 \cdot e^{2.5 \cdot \sin(10 \cdot t)} - 1.9 \cdot e^{-2 \cdot t} \tag{3}$$



Wykres 5.1. Wizualizacja wyznaczania całki nieoznaczonej metodą prostokątów
Źródło: opracowanie własne

Skrypt realizujący wyznaczenie całki oznaczonej z funkcji (3) w przedziale (0,1) zapisano poniżej.

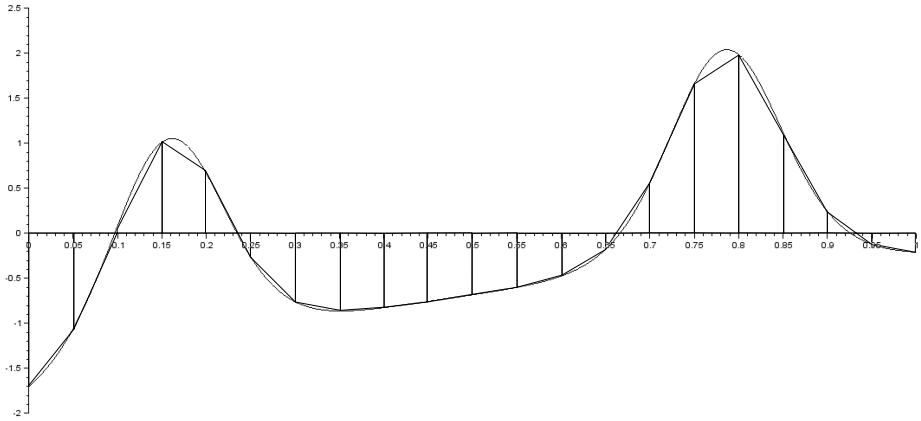
```
function y=fuct(t)
    y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)
endfunction
a=0;
b=1;
n=20;
S=0;
for k=1:n
    S=S+((b-a)/n)*fuct(k*((b-a)/n))
end
disp(S);
```

Całka oznaczona wyznaczona za pomocą metody prostokątów w granicach (0,1) wynosi 0.0294990, przy podziale na 20 przedziałów.

5.1.2. Metoda trapezów

Ze względu na duży błąd odwzorowania pola pod krzywą funkcji przez prostokąty stosowane w metodzie prostokątów opracowano metodę trapezów. Krzywa funkcji podcałkowej aproksymowana jest odcinakami prostej, a wartość całki sumą pól trapezów, jak pokazano na wykresie 5.2. Za wysokość trapezu przyjmuje się długość kroku obliczeń dx, a długości podstaw za wartości funkcji na krańcach kroku obliczeniowego. Wartości poszczególnych punktów podziału obliczeń wyznacza się na podstawie zależności (4).

$$t_i = a + i \cdot \frac{b-a}{n} \tag{4}$$



Wykres 5.2. Wizualizacja wyznaczania całki nieoznaczonej metodą trapezów
Źródło: opracowanie własne

Długość kroku w sposób analogiczny jak metodzie prostokątów (1). W efekcie równanie opisujące metodę wyznaczania całki oznaczonej metodą trapezów przyjmuje postać (5).

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\left[\sum_{i=1}^{n-1} f\left(a + i \cdot \frac{b-a}{n}\right) + \frac{f(a)+f(b)}{2}\right] \tag{5}$$

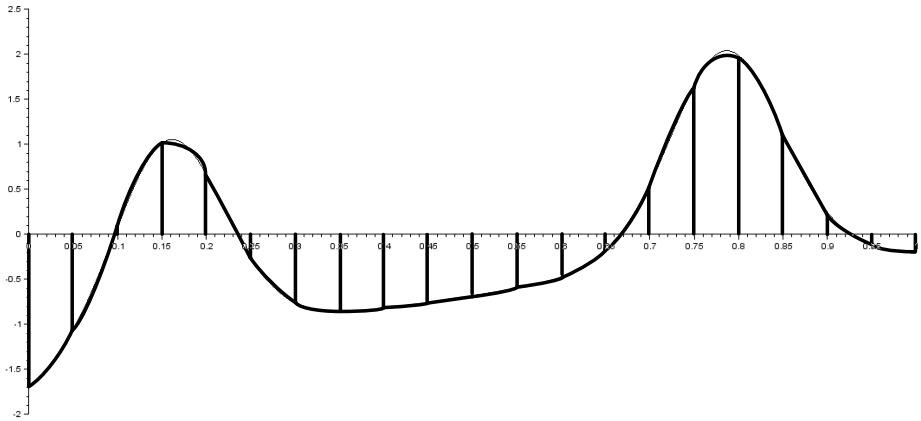
Skrypt realizujący wyznaczenie całki oznaczonej z funkcji (3) w przedziale (0,1) metodą trapezów zapisano poniżej.

```
function y=fuct(t)
    y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)
endfunction
a=0;
b=1;
n=20;
S=0;
for k=1:n-1
    S=S+fuct(a+k*(b-a)/n)
end
S=(S+(fuct(a)+fuct(b))/2)*((b-a)/n)
disp(S);
```

Całka oznaczona wyznaczona za pomocą metody trapezów w granicach (0,1) wynosi -0.0078558, przy podziale na 20 przedziałów.

5.1.3. Metoda Simpsona

Metoda Simpsona charakteryzuje się największą dokładnością. W metodzie tej krzywa funkcji podcałkowej w poszczególnych przedziałach obliczeń aproksymowana jest parabolą. Podział obszaru całkowania realizowany jest analogicznie jak w metodach prostokątów i trapezów. Dodatkowo określany jest punkt środkowy, według zależności (6) i (7).



Rys. 5.3. Wizualizacja wyznaczania całki nieoznaczonej metodą Simpsona
Źródło: opracowanie własne

$$t_i = a + i \cdot \frac{b-a}{n}, \quad i = 0,1,2...n \tag{6}$$

$$t_{m(i)} = \frac{t_{i-1} + t_i}{2}, \quad i = 1,2...n \tag{7}$$

W poszczególnych przedziałach opisanych równaniami (6) i (7) funkcja podcałkowa aproksymowana jest równaniem paraboli (8).

$$g_i(t) = A_i \cdot t^2 + B_i \cdot t + C_i, \quad t \in \langle t_{i-1}, t_i \rangle \tag{8}$$

Parametry równania wyznaczane są na podstawie trzech punktów $(t_{i-1},f(t_{i-1}))$, $(t_{m(i)},f(t_{m(i)}))$, $(t_i,f(t_i))$ z układu równań (9).

$$\begin{cases} A_i \cdot t_{i-1}^2 + B_i \cdot t_{i-1} + C_i = f_{i-1} \\ A_i \cdot t_{m(i)}^2 + B_i \cdot t_{m(i)} + C_i = f_{m(i)}, \quad i = 1,2...n \\ A_i \cdot t_i^2 + B_i \cdot t_i + C_i = f_i \end{cases} \tag{9}$$

Całka oznaczona wyznaczona za pomocą metody Simpsona w granicach (0,1) wynosi -0.0059429, przy podziale na 20 przedziałów.

Zauważyć można, jak pokazano w tabeli 5.1, że wyniki uzyskane z każdej z metod odbiegają od siebie. Dopiero znaczące zwiększenie liczby podziału przedziału całkowania sprawia, że uzyskane wyniki zaczynają być zbieżne.

Tabela 5.1 Wyniki wyznaczania całki oznaczone trzema metodami

Liczba podziału	Metoda prostokątów	Metoda trapezów	Metoda Simpsona
20	0.0294990	-0.0078558	-0.0059429
100	0.0014576	-0.0060134	-0.0059354
500	-0.0044443	-0.0059385	-0.0059354
2000	-0.0055620	-0.0059356	-0.0059354
10000	-0.0058607	-0.0059354	-0.0059354

Źródło: opracowanie własne

Analizując wyniki zestawione w tabeli 5.1 zauważyć można że rozwiązanie całki jest zbieżne do wartości -0.0059354. Jak już to było powiedziane najszybciej zbieżna jest metoda Simpsona.

5.2. Funkcje całkujące Scilab'a

W systemie Scilab zaimplementowano kilka funkcji realizujących całkowanie funkcji jednej i wielu zmiennych, oraz danych pomiarowych. Zakres tematyki ćwiczenia obejmuje zagadnienia jednej zmiennej [1],[2], wybrane funkcje zestawiono w tabeli 5.2.

Tabela 5.2 Funkcje całkujące jednej zmiennej

Składnia	Opis	Wynik funkcji (3)

`v = intg(a, b, f)`	Funkcja całkowania numerycznego, definicja funkcji wymaga podania początku i końca obszaru całkowania oraz funkcji całkowej.
	`function y=fuct(t)`
	`  y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)`
	`endfunction`
	`S1=intg(0,1,fuct)`
	`disp(S1)`
		-0.0059354

`v = inttrap(x,y)`	Funkcja całkowania danych pomiarowych metodą trapezów.
	`function y=fuct(t)`
	`  y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)`
	`endfunction`
	`t=linspace(0,1,2000)`
	`S1=inttrap(t,fuct(t))`
	`disp(S1)`
		-0.0059356

`v = intsplin(x,y)`	Funkcja całkowania danych pomiarowych za pomocą interpolacji funkcjami sklejanymi (*splines*).
	`function y=fuct(t)`
	`  y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)`
	`endfunction`
	`t=linspace(0,1,2000)`
	`S1=intsplin(t,fuct(t))`
	`disp(S1)`
		-0.0059354

Źródło: opracowanie własne

5.3. Różniczkowanie w środowisku Scilab

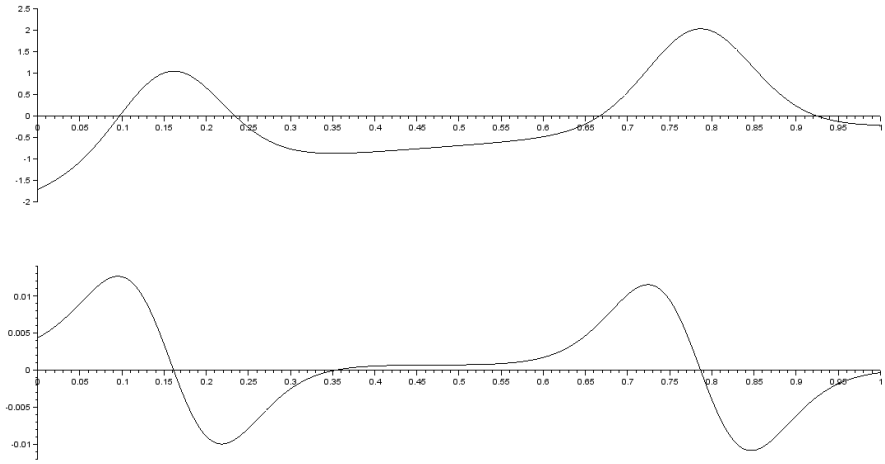
W programach obliczeniowych takich jak *Matlab* i *Scilab* zaimplementowano funkcje wyznaczające pochodne funkcji i sygnałów. W środowisku Scilab wyszczególnić można funkcję `diff` obliczającą pochodną sygnału lub funkcji *n*-tego rzędu. Składnie funkcji zapisano poniżej:

```
g=diff(f,n)
```

,gdzie: `g` - wektor pochodnej funkcji f, `f` - wektor lub funkcja, `n` - rząd pochodnej. Przykład działania funkcji diff pokazano poniżej, a na wykresie 5.4 przedstawiono przebieg funkcji (3) i jej pochodnej. Należy zwrócić uwagę, że wektor wartości pochodnej pierwszego rzędu funkcji jest o jeden element krótszy od wektora funkcji. W przypadku wyższych rzędów pochodne są krótsze od funkcji o liczbę elementów równych numerowi rzędu.

```
function y=fuct(t)
    y=0.2*exp(2.5*sin(10*t))-1.9*exp(-2*t)
endfunction
```

```
t=linspace(0,1,2000);
subplot(211);
plot2d(t,fuct(t));
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
subplot(212);
f=fuct(t);
g=numdiff(f,1);
t=linspace(0,1,1999);
plot2d(t,g);
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
```



Rys. 5.4. Przebiegi funkcji i jej pochodnej
Źródło: opracowanie własne

Zadanie

Dla funkcji podanej przez prowadzącego ćwiczenia przeanalizować przebieg zmienności w przedziale <-∞, ∞>, a w wybranym, zawężonym, przedziale obliczyć całkę oznaczoną metodami omówionymi w ćwiczeniu oraz funkcją wbudowaną. Określić optymalną liczbę podziałów obszaru całkowania dla uzyskania wszystkimi metodami wyniku zgodnego z wartością uzyskaną z funkcji wbudowanej.

6. Rozwiązywanie układów równań liniowych

Metody numerycznego rozwiązywania układów równań liniowych postaci **AX**=**B** można podzielić na dwie kategorie: metody dokładne i metody przybliżone (iteracyjne). Metody dokładne dzielą się natomiast na: eliminacyjne oraz dekompozycyjne.

6.1. Metoda Gaussa

Eliminacja Gaussa należy do metod dokładnych i polega na sprowadzeniu układu równań liniowych **AX**=**B** do nowej postaci $\mathbf{A}^{(n)}\mathbf{X}=\mathbf{B}^{(n)}$, gdzie macierz $\mathbf{A}^{(n)}$ jest macierzą trójkątną górną (macierz trójkątna górna, jest to macierz, która poniżej głównej przekątnej ma same zera). Następnie za pomocą postępowania odwrotnego nazwanego również podstawieniem wstecznym oblicza się wartości rozwiązania x_i. Aby macierz **A** przekształcić w macierz trójkątną $\mathbf{A}^{(n)}$ należy wyeliminować wyrazy spod głównej przekątnej. Stąd pochodzi nazwa metody – metoda eliminacji.

Rozwiązywanie układu trzech równań liniowych

$$\begin{cases} -5a+10b-c=12 \\ 4a+5b+20c=74 \\ 10a+3b-2c=10 \end{cases},$$

należy rozpocząć od wyboru elementu podstawowego. Krok ten zdecydowania poprawia dokładność obliczeń, a polega na takich przesunięciach wierszy i ewentualnie kolumn, aby na przekątnej głównej macierzy **A** znalazły się współczynniki o największych wartościach bezwzględnych. Bardzo ważne jest też, aby na przekątnej głównej macierzy **A** nie znajdował się współczynnik o wartości 0. Jeśli taka sytuacja występuje mimo zastosowania wyboru elementu podstawowego to układ równań nie posiada jednoznacznego rozwiązania.

W przypadku powyższego układu równań przestawiano wyłącznie wiersze, co było operacją obojętną dla wyniku. Po tej czynności układ równań ma następującą postać:

$$\begin{cases} 10a+3b-2c=10 \\ -5a+10b-c=12 \\ 4a+5b+20c=74 \end{cases}.$$

Następnie należy wyeliminować pierwszą niewiadomą z drugiego i trzeciego równania odejmując stronami pierwsze równanie pomnożone przez odpowiednie współczynniki:

$$\begin{cases} 10a + 3b - 2c = 10 \\ -5a + 10b - c - (10a + 3b - 2c)\dfrac{-5}{10} = 12 - 10\dfrac{-5}{10} \\ 4a + 5b + 20c - (10a + 3b - 2c)\dfrac{4}{10} = 74 - 10\dfrac{4}{10} \end{cases}.$$

Układ równań uzyskuje następującą postać:

$$\begin{cases} 10a + 3b - 2c = 10 \\ 11{,}5b - 2c = 17 \\ 3{,}8b + 20{,}8c = 70 \end{cases}.$$

W kolejnym kroku należy wyeliminować drugą zmienna z trzeciego równania:

$$\begin{cases} 10a + 3b - 2c = 10 \\ 11{,}5b - 2c = 17 \\ 3{,}8b + 20{,}8c - (11{,}5b - 2c)\dfrac{3{,}8}{11{,}5} = 70 - 17\dfrac{3{,}8}{11{,}5} \end{cases}.$$

Uzyskano charakterystyczną postać $\mathbf{A}^{(n)}\mathbf{X}=\mathbf{B}^{(n)}$, gdzie macierz $\mathbf{A}^{(n)}$ jest macierzą trójkątną górną:

$$\begin{cases} 10a + 3b - 2c = 10 \\ 11{,}5b - 2c = 17 \\ 21{,}461c = 64{,}383 \end{cases}.$$

Ostatnią częścią algorytmu jest postępowanie odwrotne (podstawienie wsteczne). Zaczynając od ostatniego równania można wyznaczyć wartość trzeciej niewiadomej:

$$\begin{cases} 10a + 3b - 2c = 10 \\ 11{,}5b - 2c = 17 \\ c = \dfrac{64{,}383}{21{,}461} = 3 \end{cases}.$$

Uzyskana wartość służy do wyznaczenia drugiej niewiadomej na podstawie drugiego równania:

$$\begin{cases} 10a + 3b - 2c = 10 \\ b = \dfrac{17 + 2 \cdot 3}{11{,}5} = 2 \\ c = 3 \end{cases}.$$

W ostatnim kroku wyznacza się pierwszą niewiadomą przy wykorzystaniu pierwszego równania i zgromadzonych już rozwiązań:

$$\begin{cases} a = \dfrac{10 - 3 \cdot 2 + 2 \cdot 3}{10} = 1 \\ b = 2 \\ c = 3 \end{cases}.$$

Metodę Gaussa można zapisać w postaci algorytmu:

1. jeżeli zagadnienie przedstawiono w postaci macierzy współczynników **A** i wektora wyrazów wolnych **B** należy utworzyć macierz rozszerzoną [**A B**] – połączone **A** i **B**,
2. zastosować wybór elementu podstawowego,
3. z wierszy od drugiego do ostatniego wyeliminować pierwszy wyraz odejmując wyrazy pierwszego wiersza pomnożone przez odpowiedni współczynnik,
4. rozpocząć wykonanie algorytmu od kroku numer 3 dla macierzy pomniejszonej już o skrajną lewą kolumnę i górny wiersz,
5. po zakończeniu eliminacji przeprowadzić postępowanie odwrotne.

Najprostszy skrypt zapisany dla omówionego wcześniej zadania bez wykorzystania pętli wyglądać może następująco:

```
clear
clc
// wybór elementu podstawowego został wykonany ręcznie
// przygotowana macierz współczynników
A = [10, 3, -2; -5, 10, -1; 4, 5, 20];
// wektor wyrazów wolnych
B = [10; 12; 74];
// sklejenie powyższych czyli macierz rozszerzona
AB = [A B];

// pierwszy mnożnik eliminacji
L12 = AB(2, 1) / AB(1, 1);
// eliminacja z 2 równania
AB(2, :) = AB(2, :) - L12 * AB(1, :);
// drugi mnożnik eliminacji
L13 = AB(3, 1) / AB(1, 1);
// eliminacja z 3 równania
AB(3, :) = AB(3, :) - L13 * AB(1, :);

// powrót do trzeciego punktu algorytmu
// odrzucono z rozważań wiersz nr 1 i kolumnę nr 1
// mnożnik eliminacji
L23 = AB(3, 2) / AB(2, 2);
// eliminacja z 3 równania
AB(3, 2:$) = AB(3, 2:$) - L23 * AB(2, 2:$);
// eliminacja zakończona – warto obejrzeć jak wygląda AB

// postępowanie odwrotne – obliczanie rozwiązań od końca
X(3) = AB(3, 4) / AB(3, 3);
X(2) = (AB(2, 4) - AB(2, 3) * X(3)) / AB(2, 2);
X(1) = (AB(1, 4) - AB(1, 2) * X(2) - AB(1, 3) * X(3)) /
AB(1, 1);

// dla sprawdzenia, porównanie z wynikiem działania
// funkcji Scilab-a linsolve()
Xtest = linsolve(A, -B);

disp(Xtest, 'rozwiązanie wzorcowe')
disp(X, 'rozwiązanie metodą Gaussa')
```

6.2. Metoda Gaussa-Jordana

Metoda ta została zaproponowana jako ulepszenie metody Gaussa polegające na pozbyciu się drugiego etapu obliczeń czyli postępowania odwrotnego. Metoda G-J polega na sprowadzeniu układu równań **AX**=**B** do postaci $\mathbf{EX}=\mathbf{B}^{(n)}$ gdzie macierz **E** jest macierzą jednostkową.

Rozwiązywanie układu trzech równań liniowych rozpoczęto od takiego przesunięcia wierszy macierzy rozszerzonej, aby na przekątnej głównej znalazły się współczynniki o największych wartościach bezwzględnych:

$$\begin{cases} 10a+3b-2c=10 \quad /:10 \\ -5a+10b-c=12 \\ 4a+5b+20c=74 \end{cases}.$$

Sprowadzono współczynnik przy niewiadomej a w pierwszym równaniu do jedności. Następnie wyeliminowano pierwszą niewiadomą z równań nr 2 i 3:

$$\begin{cases} a+0{,}3b-0{,}2c=1 \\ -5a+10b-c-(-5)(a+0{,}3b-0{,}2c)=12-(-5)\cdot 1, \\ 4a+5b+20c-4(a+0{,}3b-0{,}2c)=74-4\cdot 1 \end{cases}$$

$$\begin{cases} a+0{,}3b-0{,}2c=1 \\ 11{,}5b-2c=17 \quad /:11{,}5 \\ 3{,}8b+20{,}8c=70 \end{cases}.$$

Sprowadzono współczynnik przy niewiadomej b w drugim równaniu do jedności i wyeliminowano drugą niewiadomą z równań nr 1 i 3:

$$\begin{cases} a+0{,}3b-0{,}2c-0{,}3(b-0{,}174c)=1-0{,}3\cdot 1{,}478 \\ b-0{,}174c=1{,}478 \\ 3{,}8b+20{,}8c-3{,}8(b-0{,}174c)=70-3{,}8\cdot 1{,}478 \end{cases},$$

$$\begin{cases} a-0{,}148c=0{,}557 \\ b-0{,}174c=1{,}478 \\ 21{,}46c=64{,}38 \quad /:21{,}46 \end{cases}.$$

Sprowadzono współczynnik przy niewiadomej c w trzecim równaniu do jedności i wyeliminowano trzecią niewiadomą z równań nr 1 i 2 uzyskując rozwiązanie:

$$\begin{cases} a - 0{,}148c - (-0{,}148)c = 0{,}557 - (-0{,}148)\cdot 3 \\ b - 0{,}174c - (-0{,}174)c = 1{,}478 - (-0{,}174)\cdot 3 \\ c = 3 \end{cases},$$

$$\begin{cases} a = 1 \\ b = 2 \\ c = 3 \end{cases}.$$

Algorytm tego postępowania można zapisać następująco:

1. jeżeli zagadnienie przedstawiono w postaci macierzy współczynników **A** i wektora wyrazów wolnych **B** należy utworzyć macierz rozszerzoną [**A B**],
2. przeprowadzić wybór elementu podstawowego,
3. sprowadzić współczynnik leżący na przekątnej głównej macierzy [**A B**] w pierwszym (aktualnie rozpatrywanym) wierszu do jedności,
4. ze wszystkich pozostałych wierszy macierzy wyeliminować pierwszą niewiadomą odejmując wyrazy pierwszego wiersza pomnożone przez odpowiednie współczynniki,
5. rozpocząć wykonanie od kroku numer 3 dla drugiego (kolejnego) wiersza macierzy [**A B**] odrzucając z obliczeń pierwszą kolumnę macierzy.

Przykładowy skrypt Scilab-a:

```
clear
clc
// przeprowadzono wcześniej wybór elementu podstawowego
A = [10, 3, -2; -5, 10, -1; 4, 5, 20];
B = [10; 12; 74];
// sklejenie powyższych czyli macierz rozszerzona
AB = [A B];

// sprowadzenie pozycji 1,1 do jedności
AB(1,:)=AB(1,:)/AB(1,1);
// eliminacja pierwszej niewiadomej z 2. równania
AB(2, :) = AB(2, :) - AB(2, 1) * AB(1, :);
// eliminacja pierwszej niewiadomej z 3. równania
AB(3, :) = AB(3, :) - AB(3, 1) * AB(1, :);

// powrót do trzeciego punktu algorytmu
// sprowadzenie pozycji 2,2 do jedności
AB(2,:)=AB(2,:)/AB(2,2);
// eliminacja drugiej niewiadomej z 1. równania
AB(1, 2:$) = AB(1, 2:$) - AB(1, 2) * AB(2, 2:$);
```

```
// eliminacja drugiej niewiadomej z 3. równania
AB(3, 2:$) = AB(3, 2:$) - AB(3, 2) * AB(2, 2:$);

// powrót do trzeciego punktu algorytmu
// sprowadzenie pozycji 3,3 do jedności
AB(3,:)=AB(3,:)/AB(3,3);
// eliminacja trzeciej niewiadomej z 1. równania
AB(1, 3:$) = AB(1, 3:$) - AB(1, 3) * AB(3, 3:$);
// eliminacja trzeciej niewiadomej z 2. równania
AB(2, 3:$) = AB(2, 3:$) - AB(2, 3) * AB(3, 3:$);

// rozwiązanie jest w ostatniej kolumnie macierzy AB
X=AB(:,4);

// dla sprawdzenia porównanie z wynikiem funkcji linsolve()
Xtest = linsolve(A, -B);

disp(Xtest, 'rozwiązanie wzorcowe')
disp(X, 'rozwiązanie metodą Gaussa-Jordana')
```

6.3. Metoda Doolittle'a

Innym sposobem rozwiązania układu równań liniowych **AX**=**B** jest rozkład LU (dekompozycja LU). W stosunku do innych metod dokładnych, metody dekompozycyjne są efektywne pod względem wykorzystania pamięci jak i pod względem niezbędnej do wykonania liczby operacji obliczeniowych.

Macierz **A** można przedstawić w postaci iloczynu:

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U},$$

gdzie: **L** to macierz trójkątna dolna, **U** to macierz trójkątna górna.
Oznaczając:

$$\mathbf{U} \cdot \mathbf{X} = \mathbf{Z},$$

układ równań można przedstawić jako:

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{L} \cdot \mathbf{U} \cdot \mathbf{X} = \mathbf{L} \cdot \mathbf{Z} = \mathbf{B}.$$

W celu rozwiązania układu **AX**=**B** należy więc:

- rozłożyć macierz **A** na iloczyn **L** i **U**,
- rozwiązać układ **LZ**=**B** w poszukiwaniu wektora Z,
- rozwiązać układ **UX**=**Z** w poszukiwaniu wektora X.

Zgodnie z algorytmem Doolittle'a w celu rozwiązania układu

$$\begin{cases} 10a+3b-2c=10 \\ -5a+10b-c=12 \\ 4a+5b+20c=74 \end{cases},$$

macierz **A** należy rozłożyć na iloczyn **LU** taki, że na przekątnej macierzy **L** znajdują się jedynki

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Pierwszym etapem jest wyznaczenie pierwszego wiersza macierzy **U**:

$$\begin{aligned} 10 &= 1 \cdot u_{11} \Rightarrow u_{11} = 10 \\ 3 &= 1 \cdot u_{12} \Rightarrow u_{12} = 3 \\ -2 &= 1 \cdot u_{13} \Rightarrow u_{13} = -2 \end{aligned} .$$

Na podstawie uzyskanych wyników

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 & 3 & -2 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

oblicza się pierwszą kolumnę macierzy **L**:

$$\begin{aligned} -5 &= l_{21} \cdot 10 \Rightarrow l_{21} = -0{,}5 \\ 4 &= l_{31} \cdot 10 \Rightarrow l_{31} = 0{,}4 \end{aligned} ,$$

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0{,}4 & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 & 3 & -2 \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}.$$

Obliczenia trzeba kontynuować dla kolejnego wiersza macierzy **U**

$$10 = -0{,}5 \cdot 3 + 1 \cdot u_{22} \;\Rightarrow\; u_{22} = 11{,}5$$
$$-1 = -0{,}5 \cdot (-2) + 1 \cdot u_{23} \;\Rightarrow\; u_{23} = -2\,,$$

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0{,}4 & l_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 & 3 & -2 \\ 0 & 11{,}5 & -2 \\ 0 & 0 & u_{33} \end{bmatrix}$$

oraz kolejnej kolumny macierzy **L**

$$5 = 0{,}4 \cdot 3 + l_{32} \cdot 11{,}5 \;\Rightarrow\; l_{32} = 0{,}33044\,,$$

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0{,}4 & 0{,}3304 & 1 \end{bmatrix} \cdot \begin{bmatrix} 10 & 3 & -2 \\ 0 & 11{,}5 & -2 \\ 0 & 0 & u_{33} \end{bmatrix},$$

aż do ostatniego wiersza macierzy **U**

$$20 = 0{,}4 \cdot (-2) + 0{,}33044 \cdot (-2) + 1 \cdot u_{33} \;\Rightarrow\; u_{33} = 21{,}4609\,.$$

Dekompozycja doprowadziła do następujących wyników:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0{,}4 & 0{,}3304 & 1 \end{bmatrix},$$

$$\mathbf{U} = \begin{bmatrix} 10 & 3 & -2 \\ 0 & 11{,}5 & -2 \\ 0 & 0 & 21{,}4609 \end{bmatrix}.$$

Rozwiązanie układu równań $\mathbf{L} \cdot \mathbf{Z} = \mathbf{B}$ jest łatwe:

$$\begin{bmatrix} 1 & 0 & 0 \\ -0{,}5 & 1 & 0 \\ 0{,}4 & 0{,}3304 & 1 \end{bmatrix} \cdot \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix},$$

$$z_1 = 10$$
$$z_2 = 12 - (-0{,}5) \cdot 10 = 17 \quad .$$
$$z_3 = 74 - 0{,}4 \cdot 10 - 0{,}3304 \cdot 17 = 64{,}3826$$

Ostatnim etapem jest rozwiązanie układu $\mathbf{U} \cdot \mathbf{X} = \mathbf{Z}$ na drodze postępowania odwrotnego:

$$\begin{bmatrix} 10 & 3 & -2 \\ 0 & 11{,}5 & -2 \\ 0 & 0 & 21{,}4609 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10 \\ 17 \\ 64{,}3826 \end{bmatrix},$$

$$x_3 = \frac{64{,}3826}{21{,}4609} = 3$$
$$x_2 = \frac{17 - (-2) \cdot 3}{11{,}5} = 2 \quad .$$
$$x_1 = \frac{10 - 3 \cdot 2 - (-2) \cdot 3}{10} = 1$$

Przykładowy skrypt Scilab-a realizujący dekompozycję LU został przedstawiony poniżej.

```
clear
clc

// macierz współczynników
A = [10, 3, -2; -5, 10, -1; 4, 5, 20];
// wektor wyrazów wolnych
B = [10; 12; 74];

// określenie stopnia macierzy A
// (liczby wierszy)
n = size(A, 'r');

// macierz jednostkowa
L = eye(n, n);

// macierz zerowa
U = zeros(n, n);
```

```
// dekompozycja LU
for i = 1:n
    // wyznaczanie wartości wiersza macierzy U
    // od przekątnej głównej w prawo
    for j = i:n
        suma = 0;
        for k = 1:i-1 do
            suma = suma + L(i,k)*U(k,j);
        end
        U(i,j) = A(i,j) - suma;
    end
    // wyznaczanie wartości kolumny macierzy L
    // poniżej głównej przekątnej
    for j = i+1:n
        suma = 0;
        for k = 1:i-1 do
            suma = suma + L (j,k)*U(k,i);
        end
        L(j,i) = (A(j,i) - suma)/U(i,i);
    end
end

// wyznaczenie wektora Z
for i = 1:n
    suma = 0;
    for j = 1:i-1
        suma = suma + L(i,j)*Z(j);
    end
    Z(i) = B(i) - suma;
end

// wyznaczenie rozwiązania X
for i = n:-1:1
    suma = 0;
    for j = i+1:n
        suma = suma + U(i,j)*X(j);
    end
    X(i) = (Z(i) - suma)/U(i,i);
end

disp(X, 'Rozwiązanie metodą Doolittle''a')
```

Inna popularna metoda dekompozycyjna, metoda Crouta różni się jedynie wyborem współczynników macierzy trójkątnych. Jedynki znajdują się na diagonali macierzy **U**

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}.$$

6.4. Metoda Choleskiego

Dekompozycja Choleskiego jest możliwa dla macierzy symetrycznej dodatnio określonej. Macierz taka może być przedstawiona w postaci iloczynu

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T$$

gdzie $\mathbf{L}$ to dolna macierz trójkątna, $\mathbf{L}^T$ to macierz $\mathbf{L}$ transponowana. Przykładowo dla macierzy stopnia trzeciego rozkład wygląda następująco:

$$\mathbf{A} == \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \cdot \begin{bmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{bmatrix}.$$

Współczynniki macierzy $\mathbf{L}$ wyznacza się wg wzoru

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

oraz

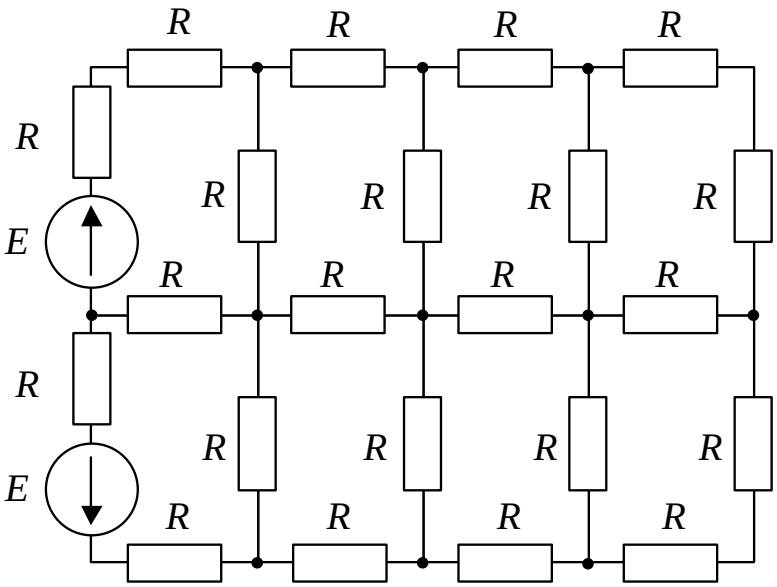
$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk}}{l_{jj}}.$$

W zależności od kolejności obliczania wartości macierzy $\mathbf{L}$ metoda nosi nazwę algorytmu Choleskiego-Banachiewicza lub algorytmu Choleskiego-Crouta.

Podobnie jak w innych metodach dekompozycyjnych po wykonaniu rozkładu macierzy $\mathbf{A}$ należy rozwiązać układ równań $\mathbf{L} \cdot \mathbf{Z} = \mathbf{B}$, a następnie $\mathbf{U} \cdot \mathbf{X} = \mathbf{Z}$ gdzie $\mathbf{U} = \mathbf{L}^T$.

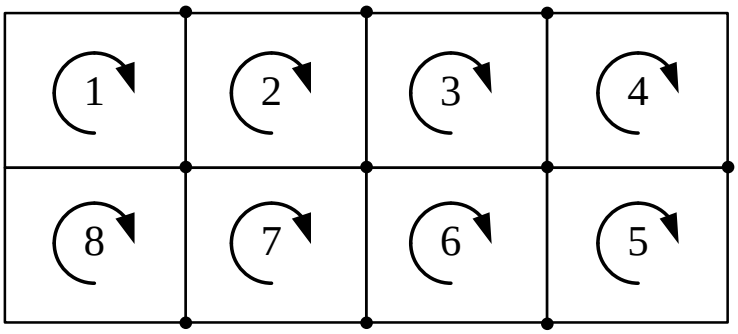
Macierze symetryczne dodatnio określone występują często przy opisie matematycznym i rozwiązywaniu problemów inżynierskich. Dobrym przykładem z zakresu elektrotechniki jest analiza obwodu prądu stałego pokazanego na

rysunku. 6.1. Celem zadania jest wyznaczenie prądów oczkowych w tym obwodzie dla danych: $R = 10\ \Omega$, $E = 100$ V.



Rysunek 6.1. Schemat obwodu prądu stałego
Źródło: opracowanie własne

Graf obwodu został pokazany na rysunku 6.2. Numeracja i oznaczenia obiegów oczek mogą być wykonane w dowolny sposób ale warto wziąć pod uwagę nakład pracy jaki będzie niezbędny do sformułowania macierzy metody oczkowej.



Rysunek 6.2. Graf obwodu z rysunku 6.1
Źródło: opracowanie własne

W przypadku grafu z rysunku 6.2 zadanie jest opisane układem równań

$$\begin{bmatrix} 4R & -R & 0 & 0 & 0 & 0 & 0 & -R \\ -R & 4R & -R & 0 & 0 & 0 & -R & 0 \\ 0 & -R & 4R & -R & 0 & -R & 0 & 0 \\ 0 & 0 & -R & 4R & -R & 0 & 0 & 0 \\ 0 & 0 & 0 & -R & 4R & -R & 0 & 0 \\ 0 & 0 & -R & 0 & -R & 4R & -R & 0 \\ 0 & -R & 0 & 0 & 0 & -R & 4R & -R \\ -R & 0 & 0 & 0 & 0 & 0 & -R & 4R \end{bmatrix} \begin{bmatrix} I_{o1} \\ I_{o2} \\ I_{o3} \\ I_{o4} \\ I_{o5} \\ I_{o6} \\ I_{o7} \\ I_{o8} \end{bmatrix} = \begin{bmatrix} E \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -E \end{bmatrix},$$

którego macierz **A** jest symetryczna i dodatnio określona.
Skrypt Scilab-a rozwiązujący tak postawiony problem przedstawiono poniżej.

```
clear
clc

// parametry zadania
n = 8; // liczba oczek
E = 100; // s. em.
R = 10; // rezystancja

// utworzenie macierzy dla metody oczkowej
// podzielono na kilka prostych etapów

// rezystancje wzajemne kolejnych oczek np. (1,2)
// czyli tworzenie wstęgi nad przekątna główną
A = diag(-R*ones(n-1,1), 1);

// rezystancje wzajemne oczek nie następujących
// po sobie oczek np. (1,8)
for i = 1:int(n/2)-1
    A(i,n-i+1) = -R;
end

// uzupełnienie macierzy symetrycznej
// poniżej głównej diagonali
A = A + A';

// rezystancje własne oczek na głównej przekątnej
A = A + eye(n, n) * 4 * R;
```

```
// utworzenie wektora wymuszeń
B(1) = E;  // pierwsze oczko
B(n) = -E; // ostatnie oczko

// dekompozycja Choleskiego dla macierzy A
// zgodnie z podanymi wcześniej wzorami
for i = 1:n
    for j = 1:n
        if i == j then
            suma = 0;
            for k = 1:i-1
                suma = suma + L(i,k)*L(i,k);
            end
            L(i,i) = sqrt(A(i,i) - suma);
        elseif i > j then
            suma = 0;
            for k = 1:j-1
                suma = suma + L(i,k)*L(j,k);
            end
            L(i,j) = (A(i,j) - suma)/L(j,j);
        end
    end
end

// wyznaczenie wektora Z
for i = 1:n
    suma = 0;
    for j = 1:i-1
        suma = suma + L(i,j)*Z(j);
    end
    Z(i) = (B(i) - suma)/L(i,i);
end

// wyznaczenie rozwiązania X
U = L';
for i = n:-1:1
    suma = 0;
    for j = i+1:n
        suma = suma + U(i,j)*X(j);
    end
    X(i) = (Z(i) - suma)/U(i,i);
end

disp(X, 'Rozwiązanie (prądy oczkowe) metodą Choleskiego')
disp(linsolve(A, -B), 'Rozwiązanie wzorcowe linsolve')
```

6.5. Metoda iteracyjna Jacobiego

Metoda Jacobiego należy do metod przybliżonych. Obliczenia są wykonywane iteracyjnie – w każdej kolejnej iteracji wyznaczane jest dokładniejsze rozwiązanie układu równań **AX**=**B**. Warunkiem zakończenia obliczeń jest uzyskanie założonej zbieżności rozwiązań – sytuacja kiedy kolejny krok nie przynosi znaczącej zmiany wyników.

Metoda sprawdza się ponieważ w praktycznych zagadnieniach często nie jest potrzebny dokładny wynik – wystarczy odpowiednio dobre jego przybliżenie. W przypadku dużych układów równań algorytmy iteracyjne są wydajniejsze od metod dokładnych.

Schemat postępowania jest następujący:

1) należy sprawdzić czy macierz główna **A** nie jest osobliwa i czy układ jest dobrze uwarunkowany tzn. czy są szanse na zbieżność rozwiązania, w uproszczeniu – należy sprawdzić czy współczynniki na głównej przekątnej są niezerowe i mają wartości dominujące,
2) przyjąć dokładność ε z jaką będzie poszukiwane rozwiązanie układu równań,
3) macierz główną układu **A** rozłożyć na sumę macierzy **L**+**D**+**U**, gdzie **L** jest macierzą poddiagonalną, **D** – macierzą diagonalną, **U** – macierzą naddiagonalną:

$$\mathbf{A}=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}=\mathbf{L}+\mathbf{D}+\mathbf{U}=$$

$$=\begin{bmatrix} 0 & 0 & 0 \\ a_{21} & 0 & 0 \\ a_{31} & a_{32} & 0 \end{bmatrix}+\begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix}+\begin{bmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

4) do dalszych obliczeń wykorzystana będzie macierz diagonalna odwrotna $\mathbf{D}^{-1}$ którą można łatwo obliczyć wyznaczając odwrotności jej niezerowych współczynników,
5) na podstawie wzoru

$$\mathbf{X}_{k+1}=\mathbf{D}^{-1}\cdot\left[\mathbf{B}-(\mathbf{L}+\mathbf{U})\cdot\mathbf{X}_k\right]$$

należy obliczyć wektor rozwiązań **X**. Obliczenia wykonuje się iteracyjnie tzn. w każdym kolejnym kroku obliczane jest dokładniejsze przybliżenie rozwiązania. Pierwszym (czy może zerowym) przybliżeniem rozwiązania mogą być dowolne wartości podane przez użytkownika jednak najczęściej przyjmowane są wyrazy wolne układu (wektor **B**).

Punkt 5 algorytmu należy powtarzać tak długo aż spełniony zostanie jeden z warunków zatrzymania iteracji:

- wykonanie maksymalnej dozwolonej liczby iteracji – wtedy jednak rozwiązanie będzie niedokładne lub błędne,
- uzyskanie założonej zbieżności rozwiązania $\left|\mathbf{X}_{k+1}-\mathbf{X}_k\right| \le \varepsilon$ gdzie ε to przyjęta dokładność bezwzględna.

W celu zobrazowania metody można przytoczyć znany z poprzednich rozdziałów przykład dobrze uwarunkowanego układu równań. Układ ten przedstawiono w postaci macierzowej:

$$\begin{cases} 10a+3b-2c=10 \\ -5a+10b-c=12 \\ 4a+5b+20c=74 \end{cases} \Leftrightarrow \mathbf{AX}=\mathbf{B} \Leftrightarrow \begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix}\begin{bmatrix} a \\ b \\ c \end{bmatrix}=\begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix}.$$

Macierz **A** należy rozłożyć na macierz trójkątną poddiagonalną **L**, macierz diagonalną **D** i macierz trójkątną naddiagonalną **U**

$$\mathbf{A}=\mathbf{L}+\mathbf{D}+\mathbf{U}=\begin{bmatrix} 0 & 0 & 0 \\ -5 & 0 & 0 \\ 4 & 5 & 0 \end{bmatrix}+\begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 20 \end{bmatrix}+\begin{bmatrix} 0 & 3 & -2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Następnym krokiem jest wyznaczenie macierzy diagonalnej odwrotnej $\mathbf{D}^{-1}$:

$$\mathbf{D}^{-1}=\begin{bmatrix} 0{,}1 & 0 & 0 \\ 0 & 0{,}1 & 0 \\ 0 & 0 & 0{,}05 \end{bmatrix}.$$

Jako pierwsze przybliżenie rozwiązania $\mathbf{X}_0$ przyjmuje się często wektor wyrazów wolnych **B**:

$$\mathbf{X}_0=\mathbf{B}=\begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix}.$$

Korzystając ze wzoru Jacobiego można teraz obliczyć następne przybliżenie rozwiązania $\mathbf{X}_1$:

$$\mathbf{X}_1 = \mathbf{D}^{-1} \cdot \left[\mathbf{B} - (\mathbf{L} + \mathbf{U}) \cdot \mathbf{X}_0\right] =$$
$$= \begin{bmatrix} 0{,}1 & 0 & 0 \\ 0 & 0{,}1 & 0 \\ 0 & 0 & 0{,}05 \end{bmatrix} \cdot \left\{ \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ -5 & 0 & -1 \\ 4 & 5 & 0 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} \right\} = \begin{bmatrix} 12{,}2 \\ 13{,}6 \\ -1{,}3 \end{bmatrix}.$$

Podobnie należy postępować w podczas obliczania kolejnych iteracji tej metody:

$$\mathbf{X}_2 = \mathbf{D}^{-1} \cdot \left[\mathbf{B} - (\mathbf{L} + \mathbf{U}) \cdot \mathbf{X}_1\right] =$$
$$= \begin{bmatrix} 0{,}1 & 0 & 0 \\ 0 & 0{,}1 & 0 \\ 0 & 0 & 0{,}05 \end{bmatrix} \cdot \left\{ \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ -5 & 0 & -1 \\ 4 & 5 & 0 \end{bmatrix} \cdot \begin{bmatrix} 12{,}2 \\ 13{,}6 \\ -1{,}3 \end{bmatrix} \right\} = \begin{bmatrix} -3{,}34 \\ 7{,}17 \\ -2{,}14 \end{bmatrix}$$

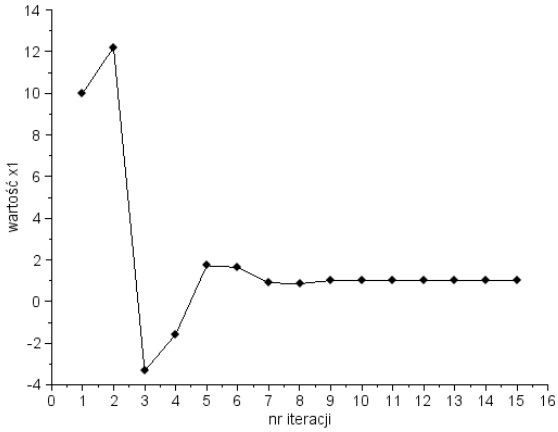
Po 14 iteracjach okazuje się, że rozwiązanie $\mathbf{X}_{14}$ (piętnaste przybliżenie) jest podobne do poprzedniego $\mathbf{X}_{13}$ na tyle, że osiągnięto przyjęty wcześniej warunek zbieżności ε=0,002:

$$\mathbf{X}_{14} = \begin{bmatrix} 1{,}0003 \\ 2{,}0009 \\ 2{,}9997 \end{bmatrix}.$$

Najprostszym przykładem praktycznym metody Jacobiego może być poniższy skrypt.

```
clear
clc
// macierz współczynników A
A = [10, 3, -2; -5, 10, -1; 4, 5, 20];
// wektor wyrazów wolnych B
B = [10; 12; 74];

// macierz poddiagonalna L
L = tril(A,-1);
// macierz naddiagonalna U
U = triu(A,1);
// współczynniki na głównej przekątnej macierzy A
```

```
listaD = diag(A);
// odwrotności tych współczynników
lista_invD = listaD^(-1);
// utworzenie macierzy diagonalnej odwrotnej
iD = diag(lista_invD);

// warunek zbieżności rozwiązania
eps = 0.002;
// maksymalna liczba iteracji
nmax = 100;

// pierwsze przybliżenie czyli B
n = 1;
X(:,n) = B;

// pętla główna programu
while n < nmax do
    n = n + 1;
    // kolejne przybliżenia
    X(:,n) = iD * (B - (L + U) * X(:,n-1));
    // sprawdzenie zbieżności rozwiązania
    if abs(X(:,n) - X(:,n-1)) <= eps then
        break
    end
end

disp(n - 1, 'Liczba iteracji');
disp(X(:,n), 'Przybliżone rozwiązanie układu równań');
```



Rysunek 6.3. Wykres wartości pierwszej niewiadomej x_1 w kolejnych iteracjach metody Jacobiego
Źródło: opracowanie własne

6.6. Metoda iteracyjna Gaussa-Seidla

Metoda Gaussa-Seidla należy do algorytmów przybliżonych rozwiązywania układów równań liniowych i jest w koncepcji podobna do algorytmu Jacobiego. Obliczenia iteracyjne realizowane są według innego równania macierzowego:

$$\mathbf{X}_{k+1} = (\mathbf{L} + \mathbf{D})^{-1} \cdot (\mathbf{B} - \mathbf{U} \cdot \mathbf{X}_k).$$

Cechą charakterystyczną tej metody jest wykorzystywanie do obliczenia następnego przybliżenia wyników poprzedniej iteracji łącznie z dostępnymi już wynikami aktualnego kroku. Do zalet należy znacząco szybsza w porównaniu z algorytmem Jacobiego zbieżność rozwiązań. Wadą jest podwyższony koszt jaki trzeba ponieść przy wyznaczaniu macierzy odwrotnej $(\mathbf{L}+\mathbf{D})^{-1}$.

Schemat postępowania jest następujący:

1) należy sprawdzić czy macierz główna **A** nie jest osobliwa i czy układ jest dobrze uwarunkowany tzn. czy są szanse na zbieżność rozwiązania, w uproszczeniu – należy sprawdzić czy współczynniki na głównej przekątnej są niezerowe i mają wartości dominujące,
2) przyjąć dokładność ε z jaką będzie poszukiwane rozwiązanie układu równań,
3) macierz główną układu **A** rozłożyć na sumę macierzy **(L+D)+U**, gdzie **(L+D)** – macierz trójkątna dolna, **U** – macierzą naddiagonalną:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = (\mathbf{L} + \mathbf{D}) + \mathbf{U} = \begin{bmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} \\ 0 & 0 & a_{23} \\ 0 & 0 & 0 \end{bmatrix}$$

4) do dalszych obliczeń wykorzystana będzie macierz trójkątna dolna odwrotna $(\mathbf{L}+\mathbf{D})^{-1}$ którą można wyznaczyć stosunkowo niskim kosztem w oparciu o jeden ze znanych algorytmów,
5) na podstawie wzoru

$$\mathbf{X}_{k+1} = (\mathbf{L} + \mathbf{D})^{-1} \cdot (\mathbf{B} - \mathbf{U} \cdot \mathbf{X}_k)$$

należy obliczać elementy wektora rozwiązań **X**. Obliczenia wykonuje się iteracyjnie tzn. w każdym kolejnym kroku obliczane jest dokładniejsze przybliżenie rozwiązania. Obliczając elementy kolejnego przybliżenia wykorzystuje się wyniki poprzedniej iteracji oraz wszystkie dostępne już wartości kolejnej. Pierwszym przybliżeniem rozwiązania mogą być dowolne wartości podane przez użytkownika jednak najczęściej przyjmowane są wyrazy wolne układu (wektor **B**).

Punkt 5 algorytmu należy powtarzać tak długo, aż spełniony zostanie jeden z warunków zatrzymania iteracji:

- wykonanie maksymalnej dozwolonej liczby iteracji – wtedy jednak rozwiązanie będzie niedokładne lub błędne,
- uzyskanie założonej zbieżności rozwiązania $\left|\mathbf{X}_{k+1} - \mathbf{X}_k\right| \le \varepsilon$ gdzie ε to przyjęta dokładność bezwzględna.

Dobrze uwarunkowany układ równań znany z poprzednich rozdziałów należy przedstawić w postaci macierzowej:

$$\begin{bmatrix} 10 & 3 & -2 \\ -5 & 10 & -1 \\ 4 & 5 & 20 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix}.$$

Macierz **A** należy rozłożyć na macierz trójkątną dolną (**L**+**D**) oraz macierz trójkątną naddiagonalną **U**

$$\mathbf{A} = (\mathbf{L} + \mathbf{D}) + \mathbf{U} = \begin{bmatrix} 10 & 0 & 0 \\ -5 & 10 & 0 \\ 4 & 5 & 20 \end{bmatrix} + \begin{bmatrix} 0 & 3 & -2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}.$$

Następnym krokiem jest wyznaczenie macierzy trójkątnej dolnej odwrotnej $(\mathbf{L}+\mathbf{D})^{-1}$:

$$(\mathbf{L} + \mathbf{D})^{-1} = \begin{bmatrix} 0{,}1 & 0 & 0 \\ 0.05 & 0{,}1 & 0 \\ -0{,}0325 & -0{,}025 & 0{,}05 \end{bmatrix}.$$

Jako pierwsze przybliżenie rozwiązania **X** przyjąć można wektor wyrazów wolnych **B**:

$$\mathbf{X} = \mathbf{B} = \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix}.$$

Korzystając ze wzoru tej metody można teraz obliczać element po elemencie następne przybliżenie rozwiązania:

$$x_1 = \begin{bmatrix} 0{,}1 & 0 & 0 \end{bmatrix} \cdot \left\{ \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} \right\} = 12{,}2\,;$$

$$x_2 = \begin{bmatrix} 0{,}05 & 0{,}1 & 0 \end{bmatrix} \cdot \left\{ \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 12{,}2 \\ 12 \\ 74 \end{bmatrix} \right\} = 14{,}7\,;$$

$$x_3 = \begin{bmatrix} -0{,}0325 & -0{,}025 & 0{,}05 \end{bmatrix} \cdot \left\{ \begin{bmatrix} 10 \\ 12 \\ 74 \end{bmatrix} - \begin{bmatrix} 0 & 3 & -2 \\ 0 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 12{,}2 \\ 14{,}7 \\ 74 \end{bmatrix} \right\} = -2{,}15\,.$$

Następnie należy powtarzać ten proces i już po 8 iteracjach okazuje się, że rozwiązanie **X** w dziewiątym przybliżeniu jest podobne do poprzedniego na tyle, że osiągnięto przyjęty warunek zbieżności ε=0,002:

$$\mathbf{X} = \begin{bmatrix} 0{,}9999 \\ 1{,}9999 \\ 2{,}9999 \end{bmatrix}.$$

Praktycznym przykładem metody G-S może być poniższy skrypt:

```
clear
clc
// macierz współczynników
A = [10, 3, -2; -5, 10, -1; 4, 5, 20];
// wektor wyrazów wolnych
B = [10; 12; 74];

// macierz trójkątna dolna L+D
LD = tril(A);
// macierz naddiagonalna
U = triu(A,1);
// utworzenie macierzy L+D odwrotnej
// tym razem z wykorzystaniem algorytmów Scilab-a
```

```
iLD = LD^(-1);

// warunek zbieżności rozwiązania
eps = 0.002;
// dopuszczalna liczba iteracji
nmax = 100;

// początkowe przybliżenie rozwiązania
n=1;
X=B;
while n < nmax do
    // zapamiętanie poprzedniego przybliżenia
    pX=X;
    // zliczanie iteracji
    n = n + 1;
    // obliczanie nowego przybliżenia element po elemencie
    // z wykorzystaniem wszystkich dostępnych wyników
    for i=1:3 do
        X(i) = iLD(i,:) * (B - U * X);
    end
    // sprawdzenie zbieżności
    if abs(X - pX) <= eps then
        break
    end
end

disp(n-1, 'Liczba iteracji');
disp(X, 'Przybliżone rozwiązanie układu równań');
```

Zadanie

Wyznaczyć prądy oczkowe w obwodzie drabinkowym prądu stałego zdefiniowanym przez prowadzącego zajęcia rozwiązując układ równań oczkowych trzema sposobami:

a) wbudowaną funkcją **linsolve**() programu Scilab,
b) jedną metodą dokładną (Gaussa, Gaussa-Jordana),
c) jedną metodą iteracyjną (Jacobiego, Gaussa-Seidla) dla dwóch podanych warunków zbieżności (przy założonej dokładności bezwzględnej na poziomie 1 mA oraz 1 μA).

Wskazówki

Współczynniki układu równań należy wyznaczyć na podstawie schematu obwodu elektrycznego. Należy unikać ręcznego wprowadzania wartości – należy

to robić używając odpowiednich funkcji Scilab-a. Bardzo przydatne będą funkcje **diag**(), **ones**() i inne z tej grupy:

```
n = 3;
R = 10;
A = diag(4 * R * ones(n, 1))
A  =
    40.    0.     0.
    0.     40.    0.
    0.     0.     40.
A = A - diag(R * ones(n - 1, 1),-1)
A  =
    40.    0.     0.
  - 10.    40.    0.
    0.   - 10.    40.
A = A - diag(R * ones(n - 1, 1), 1)
A  =
    40.  - 10.    0.
  - 10.    40.  - 10.
    0.   - 10.    40.
```

Zaletą zadania z obwodem drabinkowym jest fakt, że metody numerycznego rozwiązywania układów równań liniowych nie wymagają sprawdzania nieosobliwości macierzy, czy wyboru elementu podstawowego ponieważ są zawsze dobrze uwarunkowane.

Sprawozdanie

Do wydruku sprawozdania należy dołączyć nośnik zawierający wersję elektroniczną plików (skrypt obliczeniowy, pliki danych – macierze i wektory, dokument sprawozdania).

Sprawozdanie powinno zawierać:

a) pełny opis zadania (schemat z numeracją i oznaczonymi obiegami oczek, parametry elementów),
b) pełne skrypty Scilab-a (polecenia) rozwiązujące zadania,
c) zestawienia wyników uzyskanych rożnymi metodami w postaci tabeli z podaniem liczby iteracji metod przybliżonych,
d) sprawdzenie zgodności rozwiązań,
e) wnioski.

7. Szybka transformata Fouriera `fft`

Dane pomiarowe sygnałów napięciowych i prądowych często obarczone są dużym błędem, wynikającym z istnienia tak zwanego szumu. Jedną z metod wspomagających analizę sygnałów jest szybka transformata Fourier'a [1]. Metoda umożliwia wydzielenie z analizowanego sygnału częstotliwości i amplitud sygnałów składowych. Wynika to z założenia, że każdy sygnał można aproksymować złożeniem sygnałów sinusoidalnych o określonych częstotliwościach. W środowiskach obliczeniowych, takich jak *Matlab* czy *Scilab*, zostały zaimplementowane funkcje realizujące metodę szybkiej transformaty Fourier'a [3]. W programie *Scilab* jest to funkcja `fft`. Analiza sygnału funkcją `fft` umożliwia uzyskanie informacji o amplitudach i częstotliwościach sygnałów sinusoidalnych budujących analizowany sygnał. W efekcie możliwe jest łatwe odfiltrowanie szumu i uzyskanie "*czystego*" sygnału do dalszej analizy.

7.1. Funkcje szybkiej transformaty Fouriera w Scilab'ie

Podstawową funkcją analizy Fourier'owskiej jest `fft`, której podstawową składnię zapisano poniżej [2], [4],

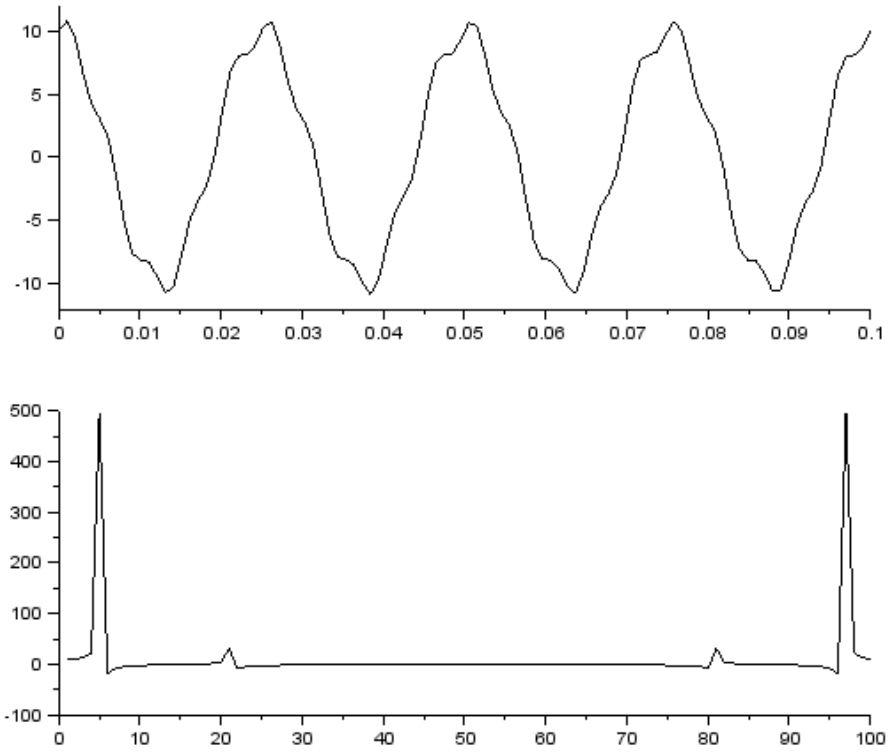
```
x=fft(a)
```

gdzie: `x` - wektor lub macierz (rzeczywisty lub zespolony) przekształcenia Fouriera,
`a` - wektor lub macierz (rzeczywisty lub zespolony) analizowanego sygnału.

Podstawowe zastosowanie funkcji szybkiej transformaty Fourier'a pokazany w skrypcie zapisanym poniżej i rysunku 7.1. W przykładzie analizowany jest sygnał z "szumem". W sygnale przetworzonym szybką transformatą Fourier'a zauważyć można dwa impulsy, pierwszy odpowiadający sygnałowi i drugi mniejszy, związany z szumem, oraz ich lustrzane odbicie. Wystarczy więc analizować połowę wektora wynikowego funkcji `fft` [4].

```
clear;clc;clf;
f=40;
N=100;
w=2*%pi*f;
t=linspace(0,4/f,N);
x = 10*cos(w*t)+5*rand(1)*sin(5*w*t);
X = fft(x);
subplot(211);
plot2d(t, x);
subplot(212);
plot2d(X);
```

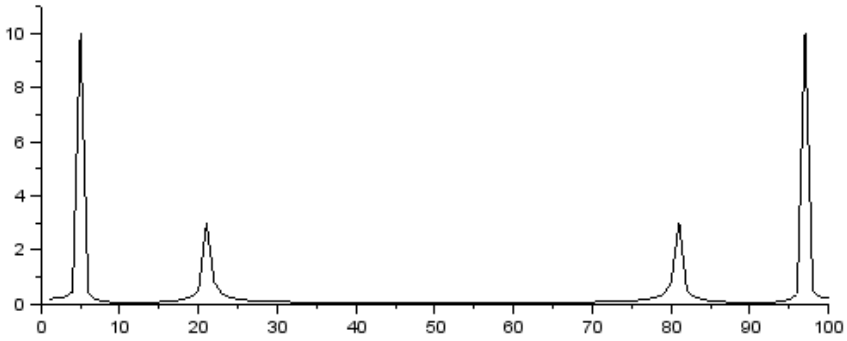
Uzyskane wyniki nie pozwalają bezpośrednio uzyskać informacji o amplitudzie i częstotliwości analizowanego sygnału. Konieczne jest przekształcenie wektora wynikowego w celu uzyskania informacji o amplitudzie oraz wygenerowanie zbieżnego z nim wektora częstotliwości w celu określenia częstotliwości analizowanego sygnału.



Rys. 7.1. Przebieg analizowanego sygnału i odpowiadającej mu transformaty
Źródło: opracowanie własne

Często wynikiem działania funkcji fft jest wektor zespolony, dlatego do dalszej analizy konieczne jest wyznaczenie modułu liczb zawartych w wektorze wynikowym. W przypadku wektora rzeczywistego, analizowanie modułu wektora także stanowi znaczące ułatwienie. W celu uzyskania informacji o amplitudzie sygnałów składowych należy elementy wektora transformaty podzielić przez połowę liczby analizowanych próbek sygnału, jak pokazano na rysunku 7.2.

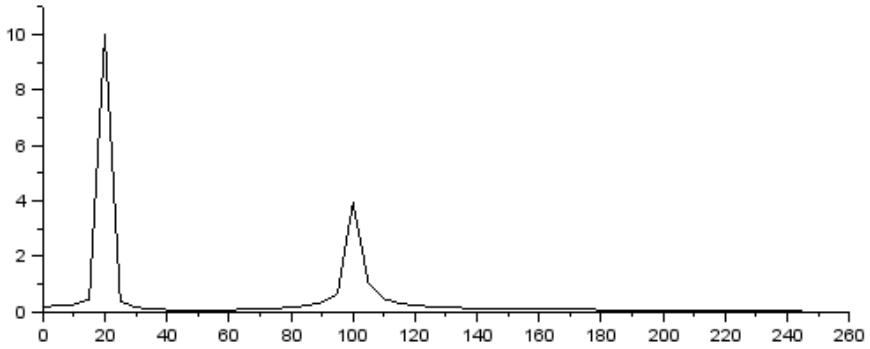
```
plot2d(abs(X)/N*2);
```



Rys. 7.2. Przebieg zmodyfikowanego sygnału transformaty
Źródło: opracowanie własne

Do dalszej analizy wystarczy wziąć połowę wektora transformaty. Kolejnym etapem jest utworzenie wektora częstotliwości zgodnego z uzyskaną transformatą, jak pokazano na rysunku 7.3. Skalę częstotliwościową wyznacza się z wykorzystaniem informacji o częstotliwości próbkowania sygnału oraz liczbie próbek w analizowanym wektorze analizowanego sygnału.

```
krok=N/(4/f);
for k=0:N/2
    ft(k)=(k)*krok/N
end;
```

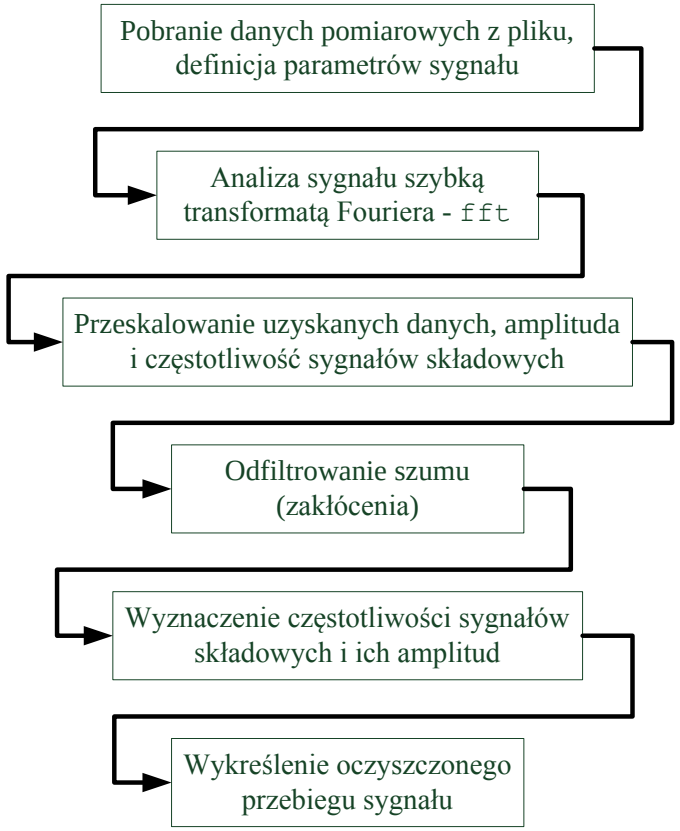


Rys. 7.3. Spektrum częstotliwościowe analizowanego sygnału
Źródło: opracowanie własne

Zauważyć można dwa piki, pierwszy o amplitudzie 10 i częstotliwości 20 Hz odpowiada analizowanemu sygnałowi, oraz drugi mniejszy o amplitudzie 4 i częstotliwości 100 Hz odpowiadający "szumowi".

7.2. Analiza sygnału pomiarowego

W środowisku Scilab'a zaimplementowano szereg narzędzi wspomagających zautomatyzowanie analizy sygnału. Cały proces analizy sygnału można podzielić na sześć etapów, jak pokazano na rysunku 7.4.



Rys. 7.4. Procedura analizy sygnału pomiarowego
Źródło: opracowanie własne

Pobranie danych pomiarowych z pliku można zrealizować za pomocą funkcji `read`,

```
X=read('nazwa.txt',k,N);
```

gdzie: `X` jest wektorem-macierzą danych z pliku zewnętrznego, `'nazwa.txt'` jest nazwą pliku z danymi pomiarowymi, `k` jest liczbą zmiennych w pliku danych, a `N` jest liczbą próbek. Należy także podać informacje o częstotliwość próbkowania oraz czasie trwania pomiarów.

Kolejne etapy to przetworzenie sygnału szybką transformatą Fourier'a, a następnie przeskalowanie uzyskanych wyników zgodnie z procedurą opisaną w rozdziale 7.1.

Odfiltrowanie szumu sprowadza się do wyzerowania wartości transformaty mniejszych od charakterystycznych pików właściwego sygnału. W składni procedury filtrującej zastosowanie mogą znaleźć funkcje wyszukiwania danych w macierzach i wektorach, `max`, `min`, `find` oraz `length`.

`X=max(A);` - wyszukuje maksymalna wartość przechowywaną w macierzy **A** i przypisuje ją do zmiennej X

`X=min(A);` - wyszukuje minimalną wartość przechowywaną w macierzy **A** i przypisuje ją do zmiennej X

`X=find(A>3);` - wyszukuje wartości z macierzy **A** większe od 3 i przypisuje ich współrzędne (indeksy) w macierzy **A** do zmiennej X

`X=lenght(A);` - przypisuje do zmiennej X liczbę równą liczbie elementów wektora **A**

Po odfiltrowaniu i określeniu parametrów szeregu sinusów możliwe jest wykreślenie "czystego" sygnału.

Zadanie

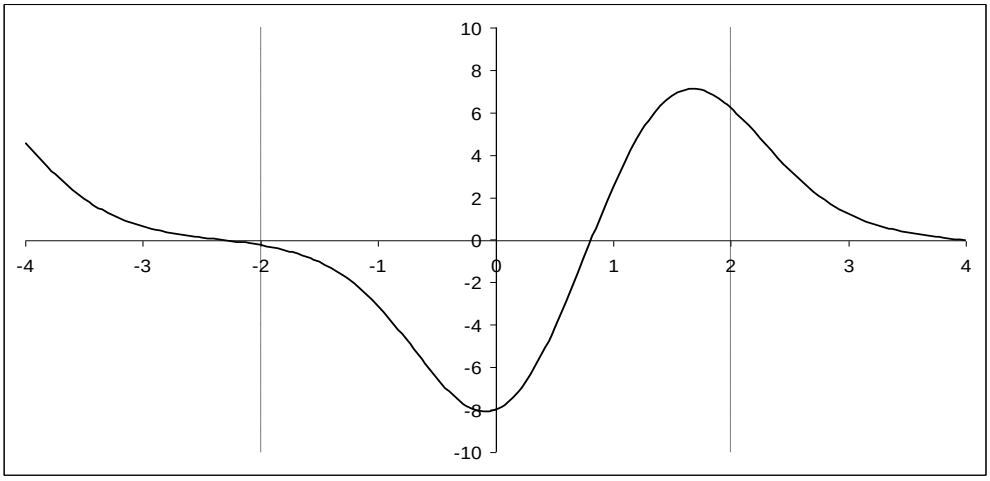
Przeanalizować sygnał podany przez prowadzącego ćwiczenia. Określić amplitudy i częstotliwości składowych sygnału, wykreślić spektrum częstotliwościowe sygnału oraz wykreślić jego odfiltrowany przebieg.

8. Wybrane metody rozwiązywania równań nieliniowych

Wiele zjawisk i urządzeń elektrycznych wymaga przy modelowaniu numerycznym rozwiązania złożonych zależności nieliniowych. Przykładem mogą obwody elektryczne zawierające elementy ferromagnetyczne [6]. W wielu sytuacja konieczne jest znalezienie rozwiązania występującego w modelu numerycznych układu kilku równań nieliniowych. Równanie takie może nie mieć żadnego rozwiązania, jedno lub wiele. Z tego względu nie jest możliwe określenie sztywnych reguł pozwalających na wyznaczenie jakiegokolwiek rozwiązania takiego równania. Opracowano kilka metod umożliwiających numeryczne znalezienie rozwiązania problemu. Zagadnienie nieliniowości jest związane z większością urządzeń elektrycznych.

8.1. Metody iteracyjne poszukiwania rozwiązania równań nieliniowych

Opracowane metody numeryczne poszukiwania rozwiązania równania nieliniowego polegają na iteracyjnym dochodzeniu do coraz dokładniejszych przybliżeń rozwiązania. W ramach ćwiczenia omówionych zostanie pięć podstawowe metody: *bisekcji*, *regula falsi*, *siecznych*, *stycznych* i *iteracji prostej* [5]. Za pomocą omówionych metod wyznaczone zostanie rozwiązanie równania $y = 2^{3\cdot\sin(x)} - 3^{2\cdot\cos(x)}$ w przedziale <-2,2>, jak pokazano na rysunku 8.1.

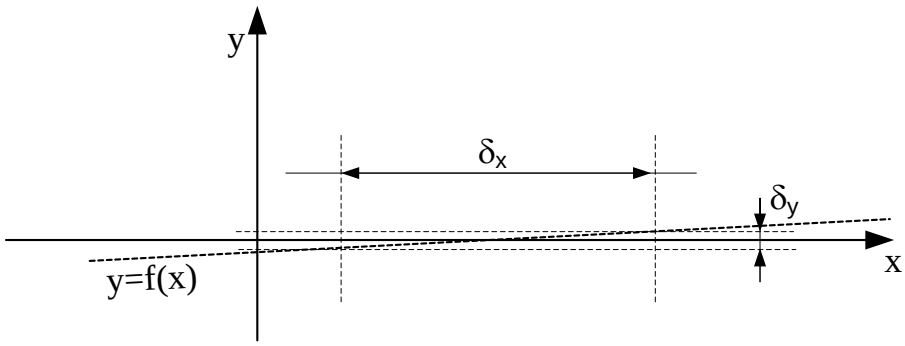


Rys. 8.1. Przebieg analizowanej funkcji z zaznaczonym przedziałem poszukiwania rozwiązania
Źródło: opracowanie własne

Analizowana funkcja jest okresowa, w całej dziedzinie liczb (-∞,∞) istnieje nieskończenie wiele rozwiązań. Dlatego konieczne jest zawężenie przedziału poszukiwania do przedziału w którym istnieje tylko jedno rozwiązanie. W zadaniu przykładowym ograniczono obszar poszukiwania do przedziału <-2,2>.

W metodach iteracyjnych wyznacza się kolejne przybliżenie rozwiązania. Wyznaczane rozwiązanie zawsze obarczone jest pewnym błędem. Ponieważ dojście do rozwiązania z dokładnością 100% mogło by wymagać wykonania dużej liczby iteracji, określa się wymaganą dokładność δ_y, po osiągnięciu której obliczenia kończą się.

W niektórych przypadkach osiągnięcie żądanej dokładności δ_y, może nastąpić w sytuacji gdy, istnieje możliwość, że wyznaczone przybliżenie x będzie obarczone dużym błędem δ_x, jak pokazano na rysunku 8.2.

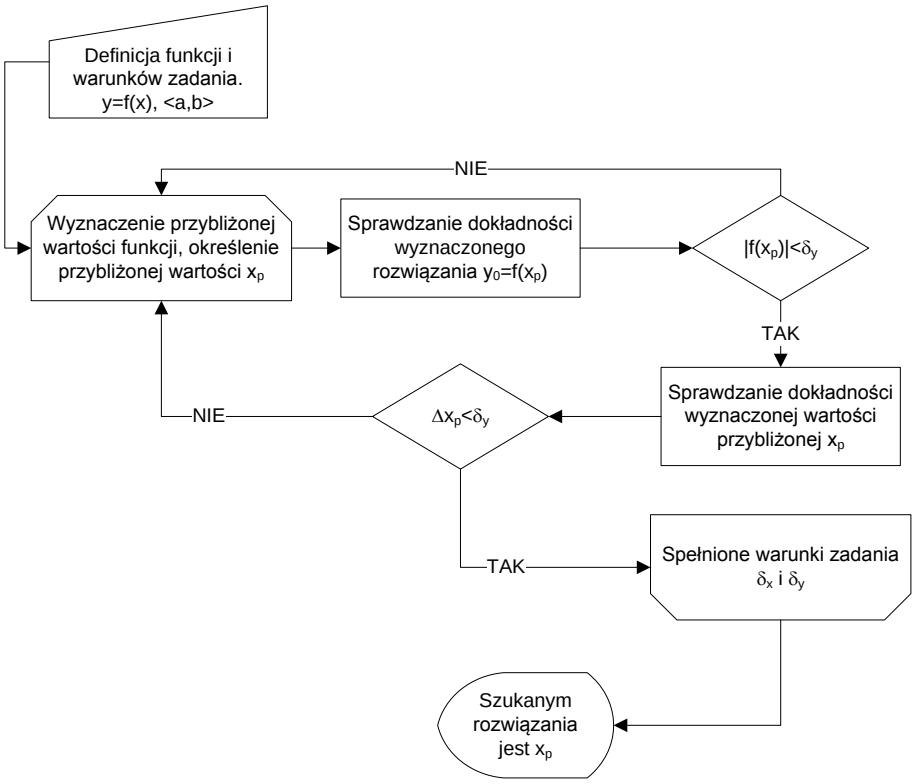


Rys. 8.2. Duży błąd δx przy dopuszczalnym błędzie δy

Źródło: opracowanie własne

Konieczne staje się więc określenie i sprawdzanie dokładności wyznaczanego rozwiązania, poprzez sprawdzanie Δx, różnicy pomiędzy kolejnymi przybliżeniami rozwiązania.

We wszystkich omawianych metodach droga dojścia do rozwiązania jest podobna i może być opisana diagramem pokazanym na rysunku 8.3.

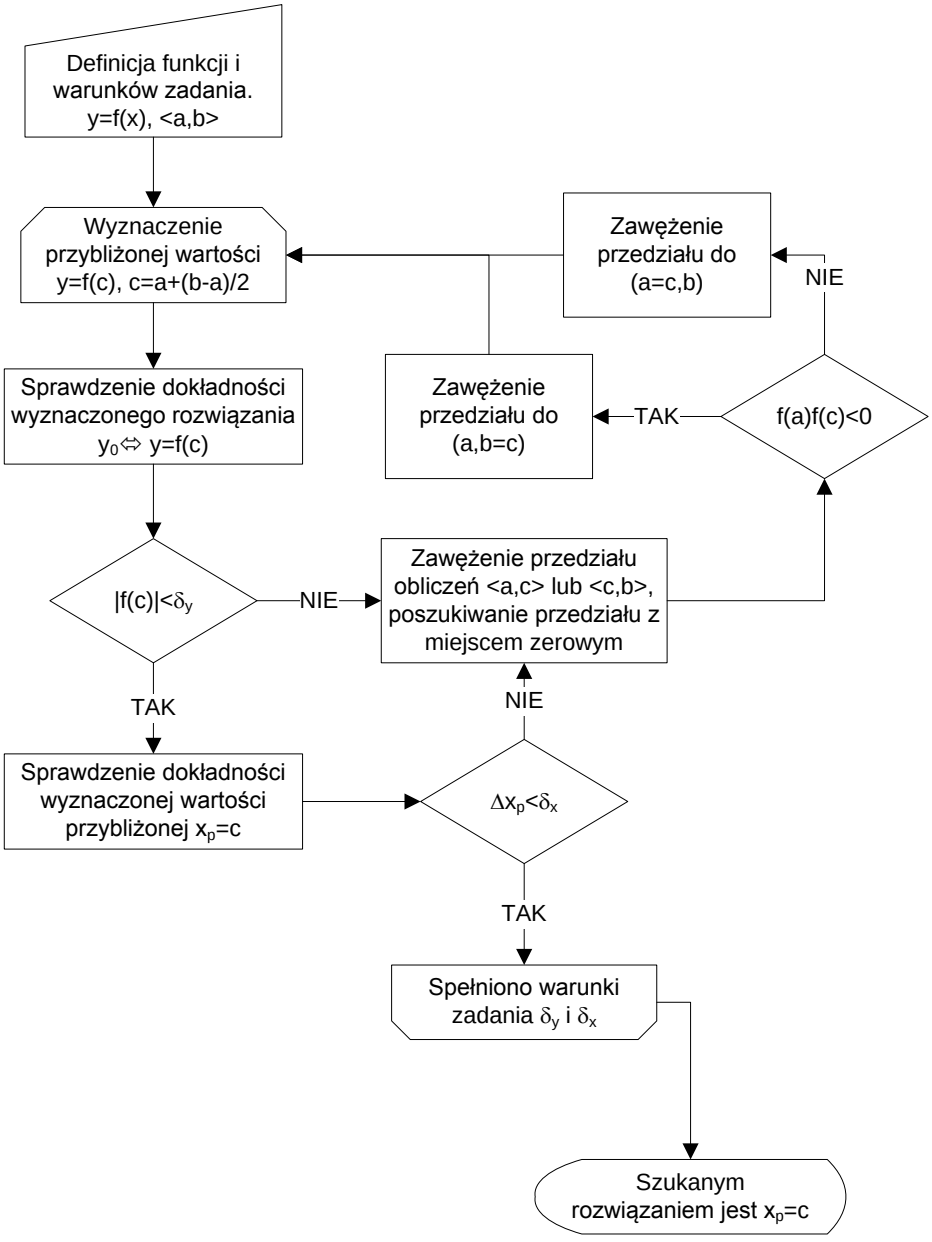


Rys. 8.3. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej
Źródło: opracowanie własne

8.1.1. Metoda Bisekcji (połowienia)

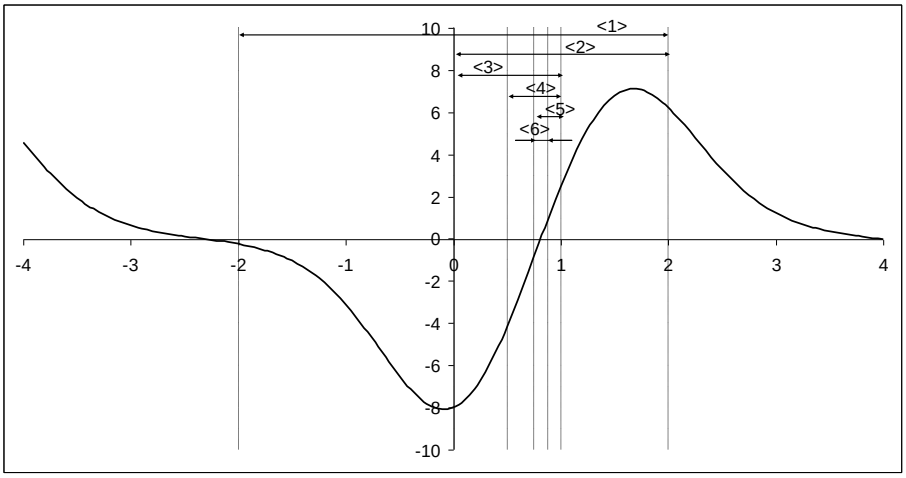
Metoda bisekcji jest najprostszą z iteracyjnych metod rozwiązywania równań nieliniowych [6]. Zapewnia ona pewną zbieżność rozwiązania, ale jest też metodą najwolniejszą, wymagającą wykonania największej liczby kroków obliczeń.

Metoda wymaga aby w badanym przedziale było tylko jedno miejsce zerowe – rozwiązanie. Metoda bisekcji polega na wyznaczaniu wartości funkcji w połowie przedziału i sprawdzaniu dla tego punktu warunków zbieżności, a następnie w przypadku niespełnienia ich na zawężaniu przedziału poszukiwań i powtarzaniu procedury. Procedurę dojścia do rozwiązania metodą bisekcji obrazuje diagram pokazany na rysunku 8.4.



Rys. 8.4. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej metodą Bisekcji
Źródło: opracowanie własne

Dochodzenie do rozwiązanie zadania zobrazowano graficznie na rysunku 8.5.



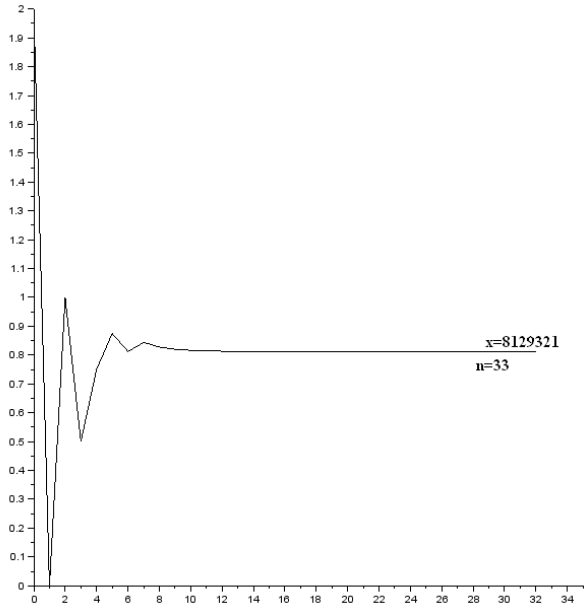
Rys. 8.5. Przebieg analizowanej funkcji wraz z zawężającymi się przedziałami poszukiwań
Źródło: opracowanie własne

Na podstawie zaprezentowanego na rysunku 8.4 diagramu napisano skrypt wyznaczający rozwiązanie przykładowego równania nieliniowego metodą bisekcji. Skrypt można podzielić na siedem części realizujące poszczególne bloki diagramu.

```
clear;clc;clf;
//definicja równania
function y=zadanie(x)
    y=2^(3*sin(x))-3^(2*cos(x));
endfunction
//definicja funkcji wyznaczającej kolejne przybliżenia
function c=szukanie(a, b)
    c=a+(b-a)/2;
endfunction
// inicjalizacja stałych
a=-2;
b=2;
delta=10^-9;
t=linspace(a,b,1000);
//wykreślanie przebiegu funkcji
subplot(121);
plot2d(t,zadanie(t));
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
//metoda bisekcji
n=1;
```

```
tn(n)=n-1;
x(n)=b;
c=b;
//główna pętla metody bisekcji
while abs(zadanie(c))>delta & abs(b-a)>delta
    n=n+1;
    tn(n)=n-1;
    c=szukanie(a,b);
    yc=zadanie(c);
    ya=zadanie(a);
    zero=ya*yc;
    if zero<0 then
        b=c;
    else
        a=c;
    end
    x(n)=c;
end
subplot(122);
plot2d(tn,x);
```

Dla przykładowej funkcji, przy dokładności δ_x i δ_y wynoszącym 10^{-9}, w zadanym przedziale (-2,2), uzyskano rozwiązanie x= 0.8129321 po 33 krokach, jak pokazano na rysunku 8.6.



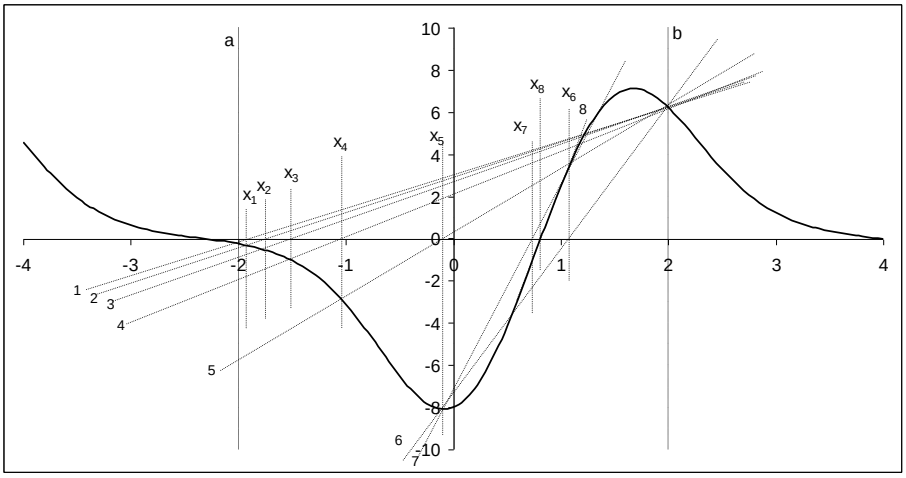
Rys. 8.6. Wykres dojścia do rozwiązania metodą Bisekcji
Źródło: opracowanie własne

8.1.2. Metoda regula-falsi

Metoda Regula Falsi (fałszywej prostej) jest iteracyjną metodą aproksymacji prostej. Metoda powstała z założenia, że każdą krzywą na dostateczne krótkim przedziale można aproksymować prostą. Kolejne przybliżenia wyznaczane są poprzez przecięcie prostej, przechodzącej przez graniczne punkty przedziału poszukiwań, z osią Ox. Przybliżenie szukanego rozwiązania zadanie wyznaczane jest z zależności 8.1.

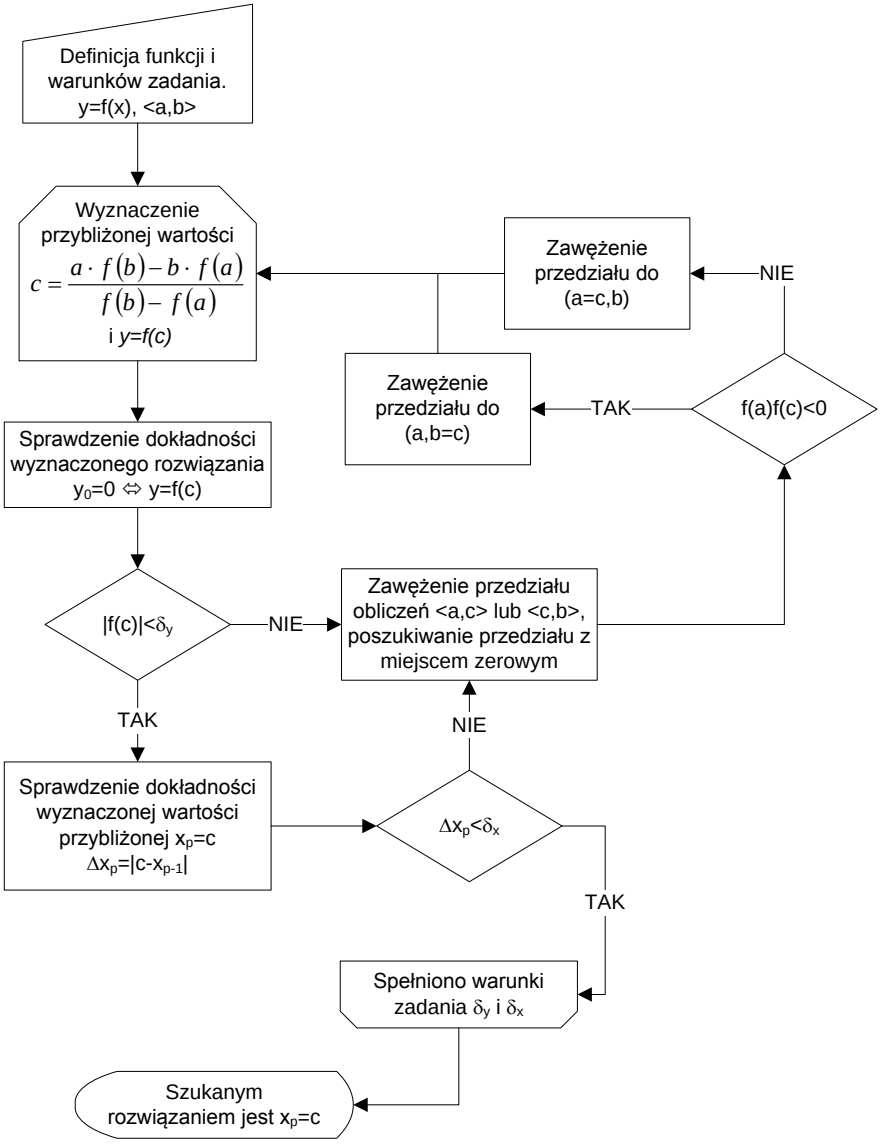
$$x_n = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)} \tag{8.1}$$

Kolejne przybliżenia niespełniające warunków zadania zawężają przedział poszukiwań, jak pokazano na rysunku 8.7.



Rys. 8.7. Poszukiwanie rozwiązania metodą regula falsi
Źródło: opracowanie własne

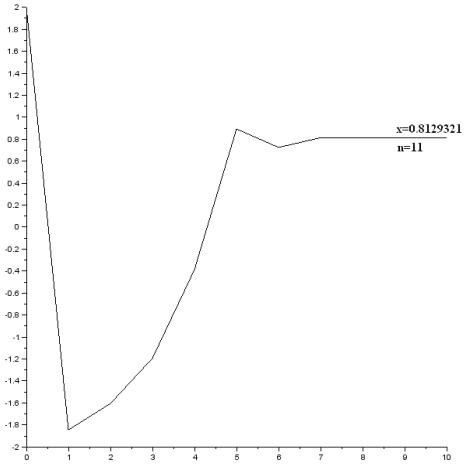
Dodatkowym warunkiem jaki muszą spełniać punkty (a,b) definiujące przedział poszukiwań jest występowanie w nim miejsca zerowego, sprawdzany jest warunek $f(a) \cdot f(b) < 0$. Jeżeli warunek nie będzie spełniony przedział poszukiwań zawęża się do dwóch ostatnich przybliżeń (x_{n-1},x_n). Analogicznie jak w metodzie bisekcji sprawdzana jest dokładność uzyskanego przybliżenia, warunkiem dwustopniowym. Analizowana jest różnica pomiędzy wartością funkcji dla wyznaczonego przybliżenia a wartością warunkową, zazwyczaj 0, oraz różnicę pomiędzy aktualnym, a poprzednim przybliżeniem. Jeżeli oba, lub jeden z warunków nie jest spełniony, wykonywany jest kolejny krok procedury poszukiwania rozwiązania. Procedurę dojścia do rozwiązania metodą Regula Falsi obrazuje diagram pokazany na rysunku 8.8.



Rys. 8.8. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej metodą Regula Falsi

Źródło: opracowanie własne

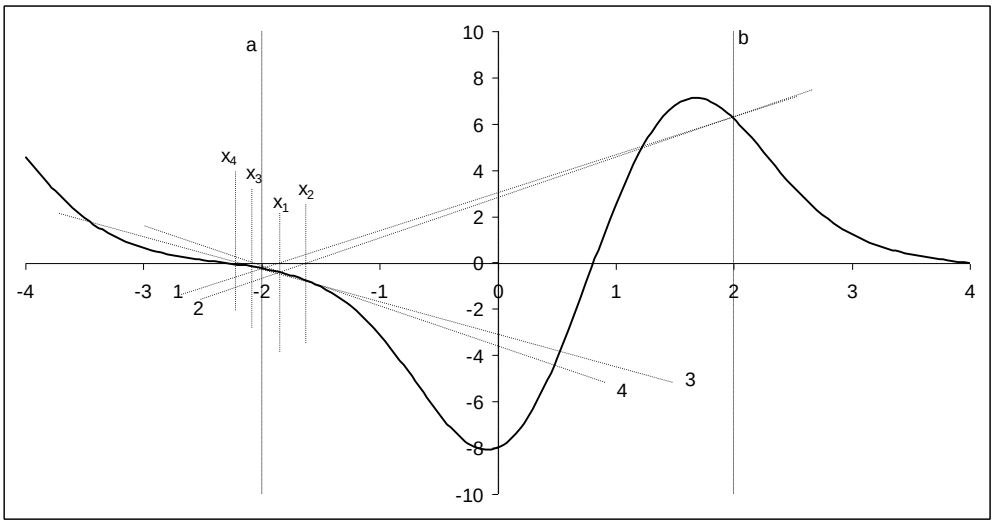
Dla przykładowej funkcji, przy dokładności δ_x i δ_y wynoszącym 10^{-9}, w zadanym przedziale (-2,2), uzyskano rozwiązanie x= 0.8129321 po 11 krokach, jak pokazano na rysunku 8.9.



Rys. 8.9. Wykres dojścia do rozwiązania metodą Regula Falsi
Źródło: opracowanie własne

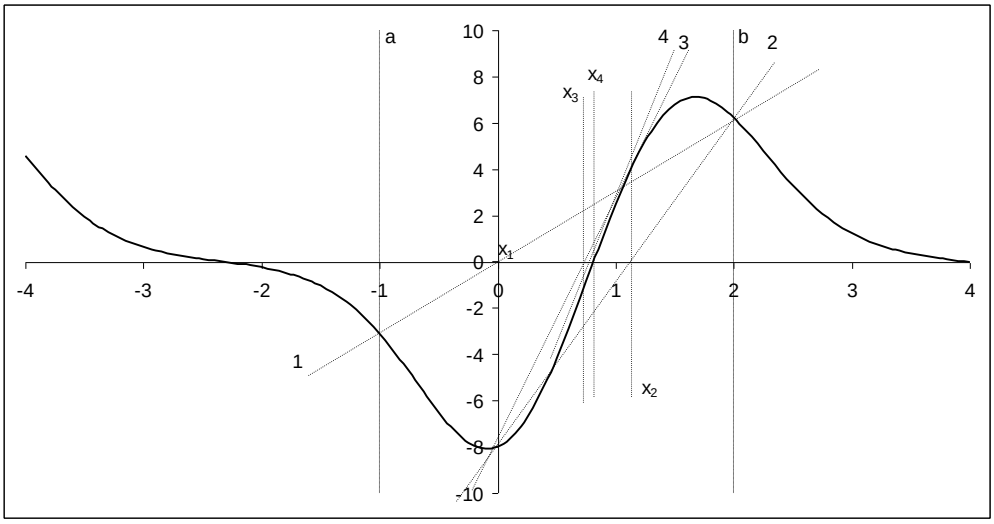
8.1.3. Metoda siecznych

Jest to kolejna metoda interpolacji liniowej, jednakże w odróżnieniu od metody Regula Falsi, granice przedziału poszukiwania rozwiązania definiowane są przez ostatnie dwa przybliżenia rozwiązania jak pokazano na rysunku 8.10. Za wartości początkowe przyjmuje się granice przedziału poszukiwania (a,b).



Rys. 8.10. Poszukiwanie rozwiązania metodą siecznych
Źródło: opracowanie własne

Metoda siecznych, w przypadku niewłaściwie zdefiniowanego przedziału poszukiwań, może nie dać rozwiązania, metoda nie jest zbieżna, albo znaleźć wartość spoza ustalonego przedziału poszukiwań, jak pokazano na rysunku 8.10. Redefiniując przedział poszukiwania, jak pokazano na rysunku 8.11 metoda znajduje rozwiązanie.

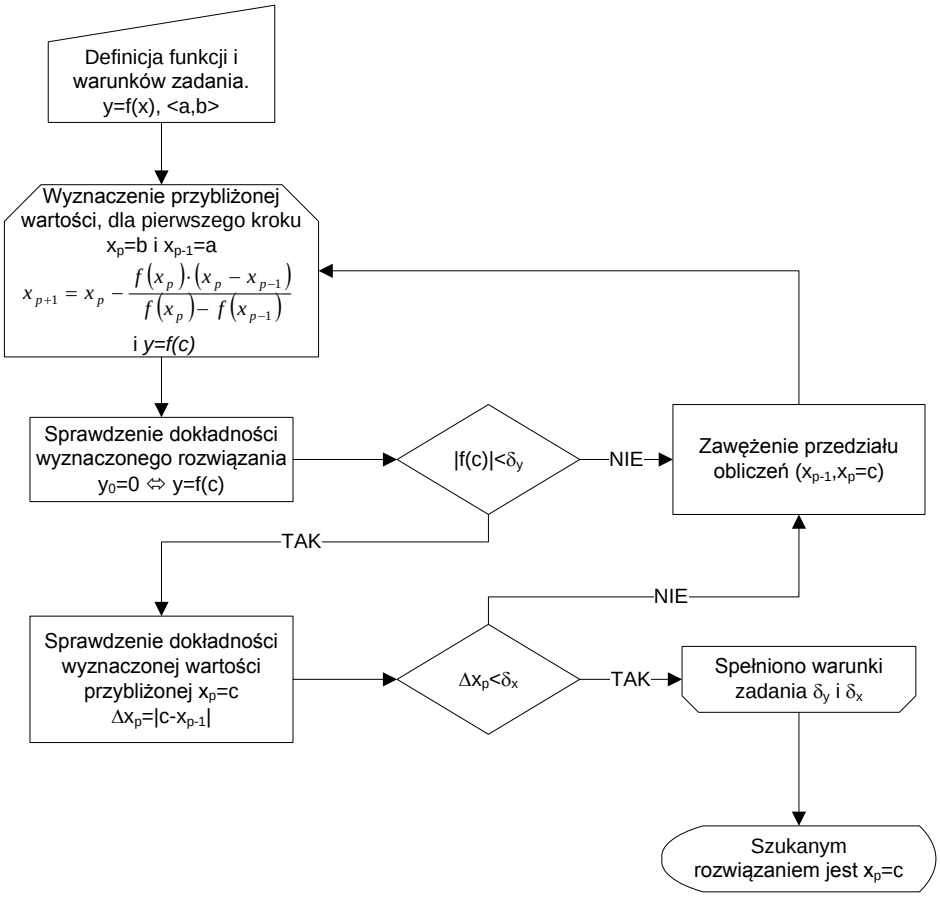


Rys. 8.11. Poszukiwanie rozwiązania metodą siecznych po zmianie przedziału poszukiwania (a,b)
Źródło: opracowanie własne

Szybkość zbieżności metody siecznych jest większa od metody regula falsi. Kolejne przybliżenia wyznaczane są z zależności opisanej równaniem 8.2.

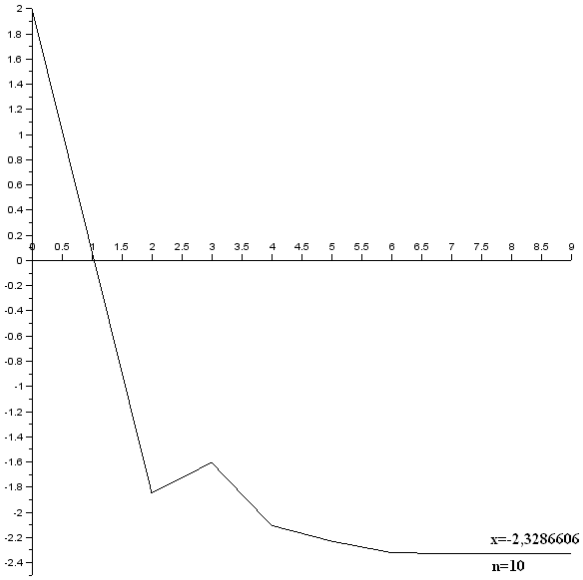
$$x_{p+1} = x_p - \frac{f(x_p)\cdot(x_p - x_{p-1})}{f(x_p) - f(x_{p-1})} \qquad 8.2$$

Procedurę dojścia do rozwiązania metodą siecznych obrazuje diagram pokazany na rysunku 8.12.

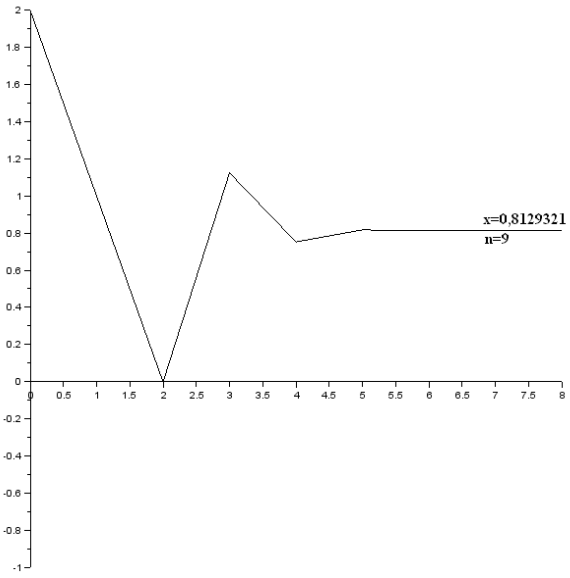


Rys. 8.12. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej metodą Siecznych
Źródło: opracowanie własne

Dla przykładowej funkcji, przy dokładności δ_x i δ_y wynoszącej 10^{-9}, w zadanym przedziale (-2,2), uzyskano rozwiązanie x= - 2.3286606 po 10 krokach, jak pokazano na rysunku 8.13. Uzyskane rozwiązanie jest poprawne, jednakże znajduje się poza założonym przedziałem poszukiwania. Konieczne było wprowadzenie zmiany do początkowego przedziału poszukiwań. Zawężając przedział występowania rozwiązania do (-1,2) uzyskano rozwiązanie x= 0.8129321 po 9 krokach, jak pokazano na rysunku 8.14, o 2 mniej jak w metodzie Regula Falsi i 24 mniej niż w metodzie bisekcji.



Rys. 8.13. Wykres dojścia do rozwiązania metodą siecznych, rozwiązanie spoza założonego przedziału (-2,2)
Źródło: opracowanie własne



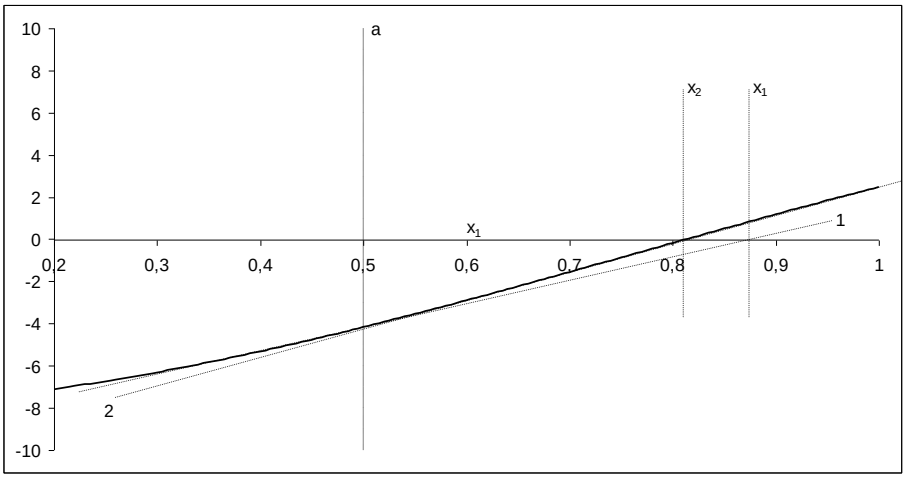
Rys. 8.14. Wykres dojścia do rozwiązania metodą siecznych
Źródło: opracowanie własne

8.1.4. Metoda stycznych

Metoda określana często jako metoda Newtona-Raphsona, o zbieżności kwadratowej. Jest ona często stosowana w tak zwanych solwerach programów obliczeniowych. Funkcja dla której poszukiwane jest rozwiązanie musi spełniać kilka warunków:

1. w przedziale poszukiwania musi posiadać dokładnie jedno rozwiązanie,
2. ma różne znaki na krańcach przedziału f(a)f(b)<0,
3. pierwsza i druga pochodna w zadanym przedziale mają stały znak.

Metoda polega na aproksymowaniu funkcji f(x) styczna prowadzoną w punkcie $P[x_{p-1},f(x_{p-1})]$, gdzie wartość przybliżenia wyznaczana jest poprzez przecięcie stycznej z osią Ox jak pokazano na rysunku 8.15. W analizowanym przypadku w celu spełnienia warunku 3 konieczne jest zawężenie przedziału poszukiwań.W praktyce istotny jest tylko punkt startowy, w analizowanym przykładzie będzie to 0.5.



Rys. 8.15. Poszukiwanie rozwiązania metodą siecznych w zawężonym przedziale poszukiwania
Źródło: opracowanie własne

Metoda wymaga wyznaczenia wartości pierwszej pochodnej funkcji w punkcie zaczepienia stycznej. Konieczne jest więc zapisanie równania pierwszej pochodnej funkcji lub jeżeli środowisko obliczeniowe pozwala wyznaczanie numeryczne wartości pochodnej w punkcie. Dla analizowanego przypadku pochodna ma postać zapisaną równanie 8.3.

$$y'(x) = 3 \cdot 2^{3\sin(x)} \cdot \ln(2) \cdot \cos(x) + 2 \cdot 3^{2\cos(x)} \cdot \ln(3) \cdot \sin(x) \tag{8.3}$$

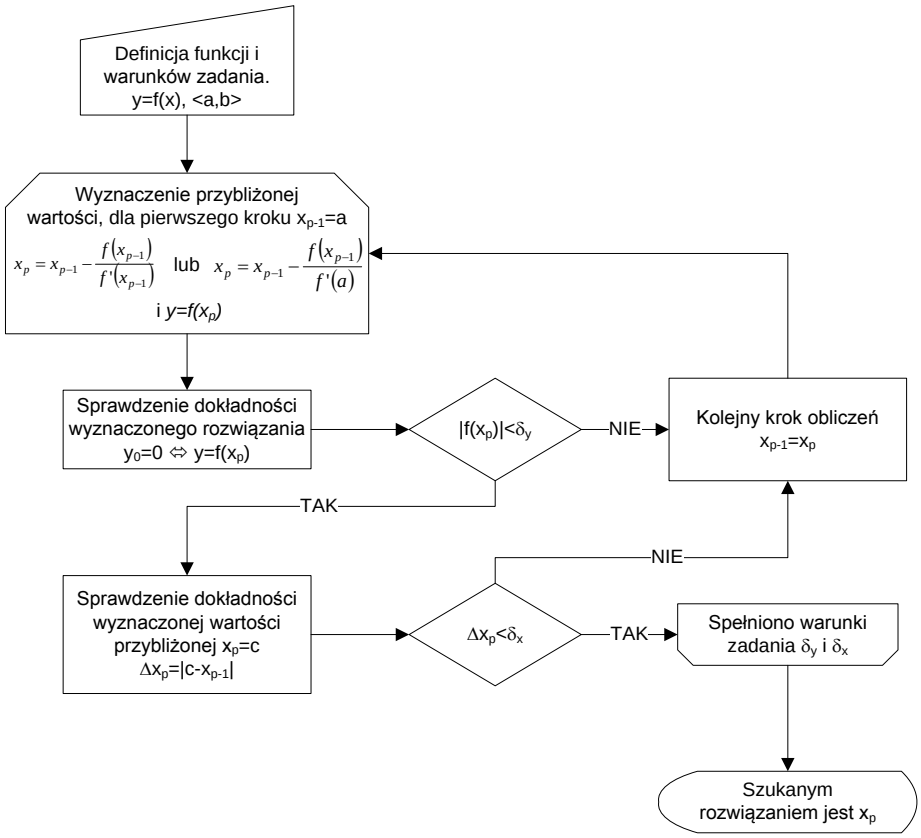
Kolejne przybliżenia rozwiązania wyznaczane są z zależności 8.4.

$$x_p = x_{p-1} - \frac{f\left(x_{p-1}\right)}{f'\left(x_{p-1}\right)} \tag{8.4}$$

Przy założeniu że f `($x_{p-1}$)≈f `(a) kolejne przybliżenia można wyznaczyć z zależności opisanej równaniem 8.5.

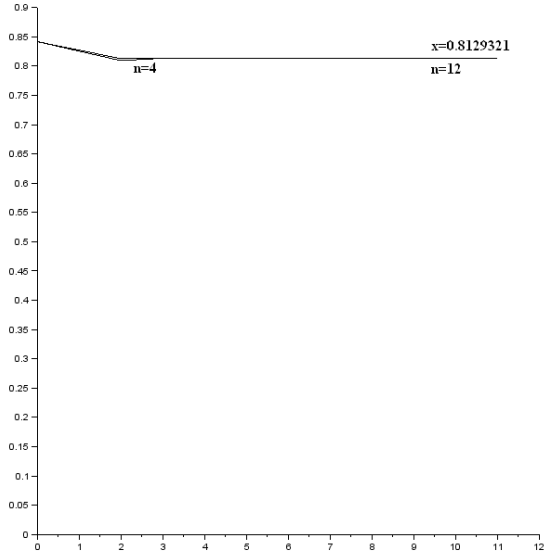
$$x_p = x_{p-1} - \frac{f\left(x_{p-1}\right)}{f'\left(a\right)} \tag{8.5}$$

Procedurę dojścia do rozwiązania metodą stycznych obrazuje diagram pokazany na rysunku 8.16.



Rys. 8.16. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej metodą Stycznych
Źródło: opracowanie własne

Dla przykładowej funkcji, przy dokładności δ_x i δ_y wynoszącym 10^{-9}, w zadanym przedziale zawężonym (0.5,2), uzyskano rozwiązanie x= 0.8129321 po 5 krokach, a z metody uproszczonej, opisanej równaniem 8.5 uzyskano rozwiązanie x= 0.8129321 po 12 krokach jak pokazano na rysunku 8.17.



Rys. 8.17. Wykres dojścia do rozwiązania metodą stycznych
Źródło: opracowanie własne

Można zauważyć, że obydwa sposoby realizacji metody stycznych dały ten sam wynik, jednakże postać uproszczona wymagała wykonania trzy razy większej liczby iteracji.

8.1.5. Metoda iteracji prostej

W metodzie iteracji prostej rozwiązanie równania f(x)=0 realizuje się poprzez wyznaczanie kolejnych wartości wyrażenia równoważnego do badanego równania według zależności 8.6.

$$x_p = g(x_{p-1}) \tag{8.6}$$

Wyrażenie równoważne musi spełnić warunek zapisany równaniem 8.7, aby ciąg kolejnych przybliżeń okazał się zbieżny do rozwiązania.

$$\left|x_{p+1} - x_p\right| < \left|x_p - x_{p-1}\right| \tag{8.7}$$

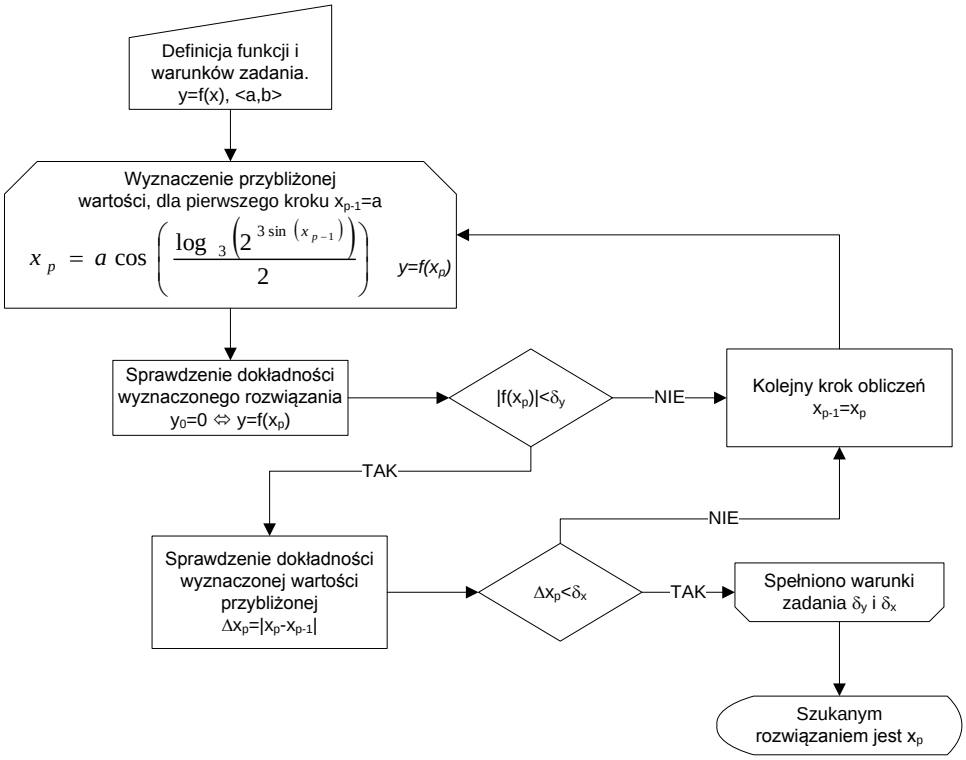
Dla analizowanego przypadku można zapisać dwie postaci wyrażeń równoważnych, jak pokazano za pomocą równań 8.8.

$$x = a\cos\left(\frac{\log_3\left(2^{3\sin(x)}\right)}{2}\right)$$
$$x = a\sin\left(\frac{\log_2\left(3^{2\cos(x)}\right)}{3}\right) \qquad 8.8$$

Warunek opisany równaniem 8.7 jest spełniony tylko dla pierwszego z równań równoważnych i ono musi zostać użyte w procedurze rozwiązania równania metodą iteracji prostej. Niestety Scilab nie posiada funkcji ani procedury umożliwiającej obliczenie logarytmu o podstawie 3. Z tego powodu konieczne jest napisanie funkcji realizującej to zadanie na podstawie wzoru 8.9.

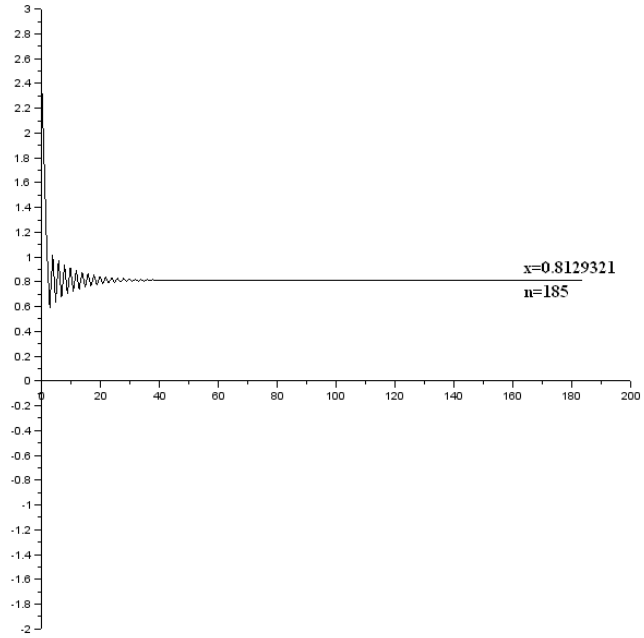
$$\log_a b = \frac{\log_c b}{\log_c a} \qquad 8.9$$

Za podstawę logarytmu we wzorze przekształcającym wygodnie jest obrać liczbę e i logarytm naturalny.



Rys. 8.18. Diagram procesu dochodzenia do rozwiązania funkcji nieliniowej metodą Iteracji prostej
Źródło: opracowanie własne

Procedurę dojścia do rozwiązania metodą iteracji prostej obrazuje diagram pokazany na rysunku 8.18. Dla przykładowej funkcji, przy dokładności δ_x i δ_y wynoszącym 10^{-9}, w zadanym przedziale (-2,2), uzyskano rozwiązanie x=0.8129321 po 185 krokach, jak pokazano na rysunku 8.19.



Rys. 8.19. Wykres dojścia do rozwiązania iteracji prostej
Źródło: opracowanie własne

Podsumowanie omówionych metod rozwiązywania równań nieliniowych zestawiono w tabeli 8.1.

Tabela 8.1 Zestawienie wyników uzyskanych z poszczególnych metod

Nazwa metody	*Bisekcji*	*Regula Falsi*	*Siecznych*	*Stycznych*	Iteracji prostej
Przedział	(-2,2)	(-2,2)	(-1,2)	(0.5,2)	(-2,2)
Liczba kroków iteracji	33	11	9	5 lub 12	185
Wynik	0.8129321	0.8129321	0.8129321	0.8129321	0.8129321

Źródło: opracowanie własne

Zauważyć można że metoda wszystkie metody zwróciły ten sam wynik. Jednakże różnią się liczbą iteracji jakich potrzebują dla uzyskania prawidłowego wyniku przy zadanych warunkach rozwiązania. Najmniej iteracji wymaga metoda stycznych, jednakże konieczne jest przy jej zastosowaniu wyznaczenie pierwszej pochodnej badanej funkcji. W zależności od przebiegu funkcji konieczne może być zawężenie przedziału poszukiwania. W niektórych przypadkach metoda może być rozbieżna i nie dać rozwiązania, albo wyznaczyć rozwiązanie, ale spoza

przedziału. Podobne sytuacja może wystąpić w metodzie siecznych, gdzie przy zbyt szerokim przedziale poszukiwań metoda może dać wynik spoza analizowanego zakresu, wymagane jest także zawężenie przedziału poszukiwań rozwiązania.

8.1.6. Funkcje wbudowane

Scilab posiada wbudowane funkcje umożliwiające rozwiązywanie równań nieliniowych jednej lub wielu zmiennych [1], [2], [3],

- `fsolve` – rozwiązuje układy równań nieliniowych,
- `lsqrsolve` – rozwiązuje układy równań nieliniowych w sytuacji gdy liczba równań jest większa od liczby niewiadomych.

8.1.6.1. Funkcja fsolve

Funkcja `fsolve` umożliwia rozwiązywanie układów równań *n*-tego stopnia. W konfiguracji minimalnej wymaga zdefiniowania dwóch parametrów wejściowych. Pierwszym jest wektor, o długości *n*, wartości początkowych poszukiwanych niewiadomych. Drugim jest wskazanie na funkcję definiującą układ równań nieliniowych. Wynikiem działania funkcji `fsolve` jest wektor rozwiązań zdefiniowanego układu równań. Rozwiązanie zadania przykładowego za pomocą funkcji `fsolve` zaprezentowano na przykładzie poniżej.

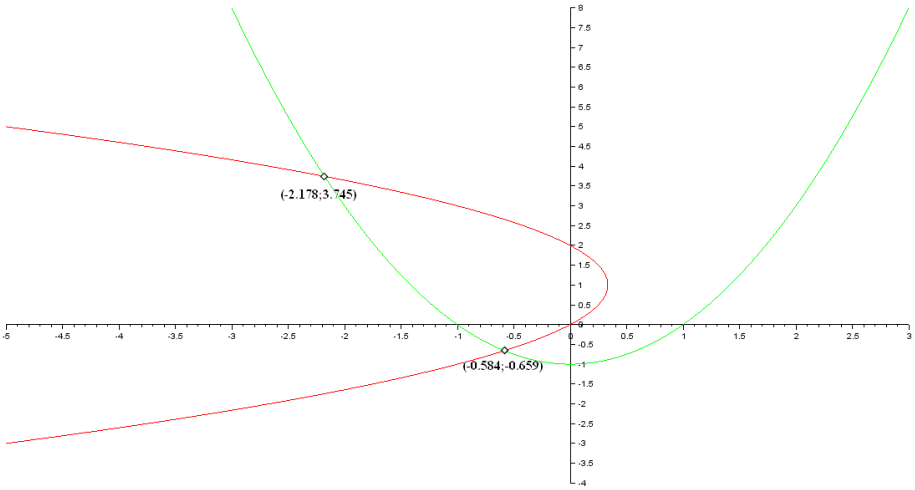
```
function y=zadanie(x)
    y=2^(3*sin(x))-3^(2*cos(x));
endfunction
x0=1;
x=fsolve(x0,zadanie);
disp(x);
```

Przykład rozwiązania układu równań nieliniowych, zapisanych równaniami 8.10 pokazano na przykładzie poniżej, wraz z rozwiązaniem graficznym pokazanym na rysunku 8.20.

$$\begin{cases} y - x^2 + 1 = 0 \\ x - \dfrac{2y - y^2}{3} = 0 \end{cases} \qquad 8.10$$

```
function [z]=zadanie(x)
    z(1)=x(2)-x(1).^2+1;
    z(2)=x(1)-(2*x(2)-x(2).^2)/3;
endfunction
x1=linspace(-3,3,101);
y1=x1.^2-1;
```

```
y2=linspace(-3,5,101);
x2=(2*y2-y2.^2)/3;
plot2d(x1,y1,3);
plot2d(x2,y2,5);
pocz=[-5;8];
roz=fsolve(pocz,zadanie)
disp(roz)
plot2d(roz(1:1),roz(2:2),-5)
pocz=[3;-4];
roz=fsolve(pocz,zadanie)
disp(roz)
plot2d(roz(1:1),roz(2:2),-5)
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
```

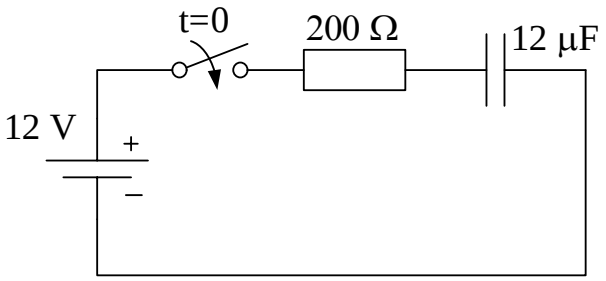


Rys. 8.20. Graficzna prezentacja rozwiązania układu równań
Źródło: opracowanie własne

8.1.6.2. Funkcja lsqrsolve

Drugą funkcją stosowaną do rozwiązywania układów równań nieliniowych jest `lsqrsolve`. Funkcja ta umożliwia rozwiązywanie układów równań *n*-tego stopnia *m* niewiadomych, gdzie *m*<*n*. Zastosowanie znajduje w wyznaczaniu parametrów funkcji aproksymujących dane pomiarowe. W takich sytuacjach danych pomiarowych, a więc i równań może być nawet kilkaset, podczas gdy parametrów równania aproksymującego może być kilka. Składnia funkcji `lsqrsolve` jest podobne do składni `fsolve`, dochodzi tylko jeden parametr obowiązkowy, zawierający informację o liczbie rozwiązywanych równań.

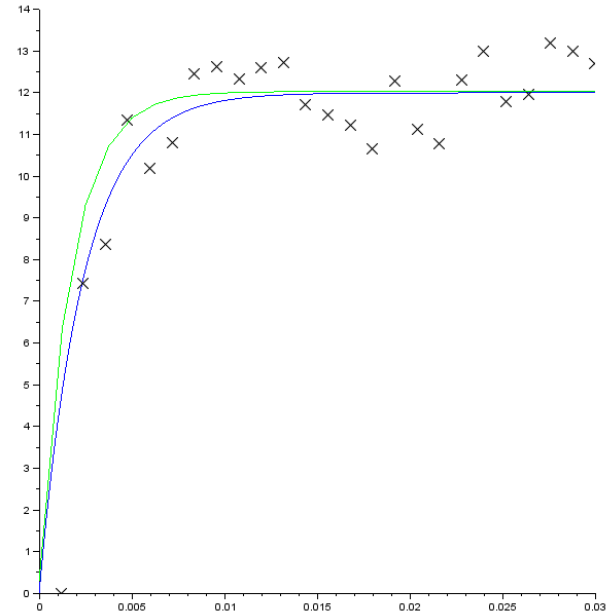
Definicja równań rozwiązywanych musi zwierać dodatkową zmienną `m` przechowującą informacje o liczbie równań nieliniowych. Zastosowanie funkcji `lsqrsolve` zostanie zaprezentowane na przykładzie wykreślania aproksymacji charakterystyk napięciowej i prądowej układu RC w stanie nieustalonym w obwodzie pokazanym na rysunku 8.21. Efektem działania programu są wykresy napięcia na kondensatorze i prądu w obwodzie, wykreślone na podstawie parametrów elementów, pomiarów oraz krzywa aproksymacyjna jak pokazano na rysunku 8.22 i 8.33. Wyniki pomiarów laboratoryjnych zostały w przykładowym skrypcie zasymulowane.



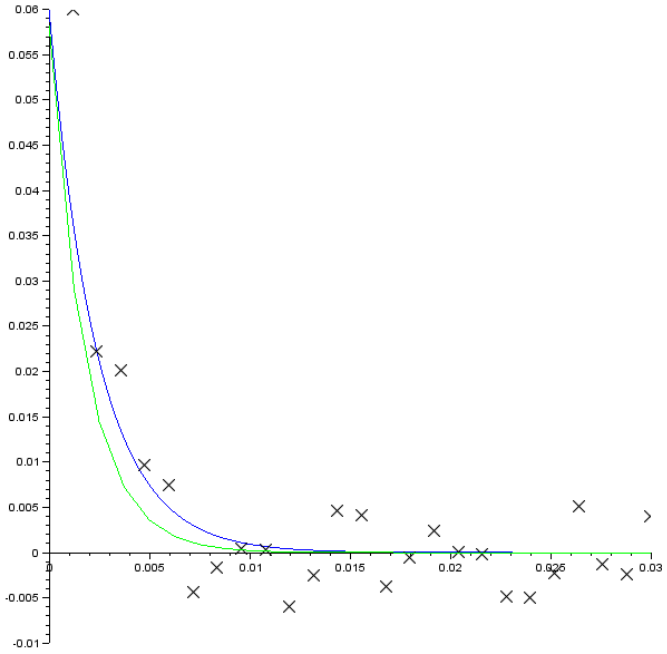
Rys. 8.21. Schemat analizowanego obwodu
Źródło: opracowanie własne

```
clear;clc;clf;
function u=napiecie(t)// funkcja przebiegu napięcia
kondensatora
    u=E-E*exp((-1/(R*C))*t);
endfunction
function i=prad(t) // funkcja przebiegu prądu
    i=(E/R)*exp((-1/(R*C))*t);
endfunction
E=12; // Parametry obwodu
R=200;
C=12*10^-6;
t=linspace(0,0.03,1000);//czas symulacji
t2=linspace(0,0.03,25)';//czas pomiarów
Uc=napiecie(t);
Ic=prad(t);
for n=1:25// symulacja pomiarów
    k(n)=t(n*40);
    Ic2(n)=Ic(40*n)+(rand(1)-0.5)*0.015;
    Uc2(n)=Uc(40*n)+(rand(1)-0.5)*3;
end
function err=eUc(par, m)// definicja funkcji aproksymacyjnej
napięcia
```

```
    err=Uc2-(par(1)-par(2)*exp(-par(3)*t2));
endfunction
function err=eIc(par, m) // definicja funkcji
aproksymacyjnej prądu
    err=Ic2-(par(1)*exp(-par(2)*t2));
endfunction
function u=aprUc(par, t)//definicja przebiegu napięcia do
wykreślenia przebiegu aproksymowanego
    u=par(1)-par(2)*exp(-par(3)*t);
endfunction
function i=aprIc(par, t) //definicja przebiegu prądu do
wykreślenia przebiegu aproksymowanego
    i=par(1)*exp(-par(2)*t);
endfunction
init_1=[0.01,0.01,0.01];//parametry poczatkowe
init_2=[0.01,0.01];
Uc2(1)=Uc(1);//definicja warunków początkowych obwodu
Ic2(1)=Ic(1);
par_1=lsqrsolve(init_1,eUc,size(t2,1));//wyznaczenie
parametrów równań aproksymacyjnych
par_2=lsqrsolve(init_2,eIc,size(t2,1));
disp(par_1);// wyświetlenie parametrów równań
aproksymacyjnych
disp(par_2);
apUc=aprUc(par_1,t2);//Wyznaczenie przebiegów
aproksymowanych
apIc=aprIc(par_2,t2);
subplot(121);
plot2d(t,Uc,2);
plot2d(t2,[Uc2 apUc],[-2,3]);
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
subplot(122);
plot2d(t,Ic,2);
plot2d(t2,[Ic2 apIc],[-2,3]);
ster_axe=gca();
ster_axe.x_location="origin";
ster_axe.y_location="origin";
```



Rys. 8.22. Przebiegi symulacyjne napięcia kondensatora w stanie nieustalonym



Rys. 8.23. Przebiegi symulacyjne prądu obwodu w stanie nieustalonym

Analitycznie wyznaczono równania opisujące badany obwód za pomocą równań 8.11, a na podstawie symulacji pomiarów wyznaczono parametry równań aproksymacyjnych 8.12.

$$\begin{cases} u_C(t) = 12 - 12 \cdot e^{-416{,}67t} \\ i_C(t) = 0{,}06 \cdot e^{-416{,}67t} \end{cases} \qquad 8.11$$

$$\begin{cases} u_C(t) = 11{,}98 - 11{,}86 \cdot e^{-883{,}8t} \\ i_C(t) = 0{,}058 \cdot e^{-671{,}1t} \end{cases} \qquad 8.12$$

Zauważyć można że jedynie wykładnik `e` odbiega znacząco od wartość wyznaczonej analitycznie. Wynika to z liniowego rozkładu pomiarów i konsekwencji małej liczby danych w obszarze przegięcia przebiegów prądu i napięcia.

Zadania

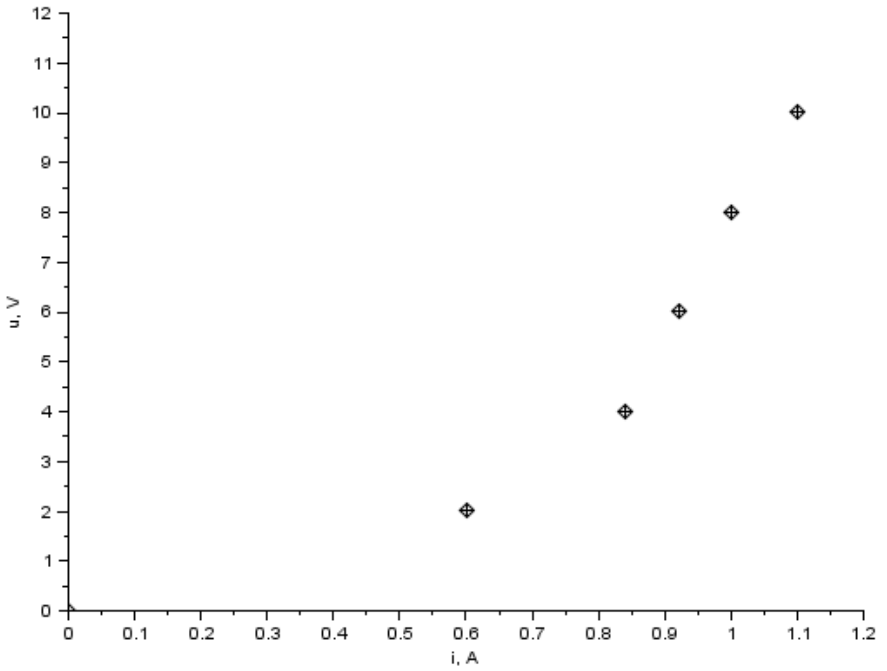
Przeprowadzić analizę numeryczną układu rezonansowego podanego przez prowadzącego ćwiczenia. W ramach ćwiczenia należy:

1. wyprowadzić równania opisujące przebiegi prądów i napięć obwodu w funkcji częstotliwości,
2. przeprowadzić symulację pomiarów, 25-30 punktów pomiarowych z błędem pomiaru na poziomie 10%,
3. wyznaczyć parametry funkcji aproksymujących przebiegi pomiarowe (funkcja `lsqrsolve`),
4. na podstawie przebiegów aproksymowanych i symulowanych określić częstotliwość rezonansową dwoma podanymi przez prowadzącego ćwiczenia metodami oraz funkcją `fsolve`,
5. wykreślić wyznaczone przebiegi, zaznaczyć graficznie i tekstowo na wykresach wyznaczone częstotliwości rezonansowe (wartość uśrednioną z symulacji i wartość uśrednioną z krzywych aproksymowanych).

9. Interpolacja i aproksymacja

Każdy, kto choć raz wykonywał serię pomiarów w laboratorium stanął przed problemem interpolacji lub aproksymacji uzyskanych wyników.

Na wykresie 9.1 przedstawiono w formie graficznej wyniki pomiarów napięcia i natężenia prądu potrzebne do wyznaczania charakterystyki pewnego nieliniowego elementu elektrycznego.

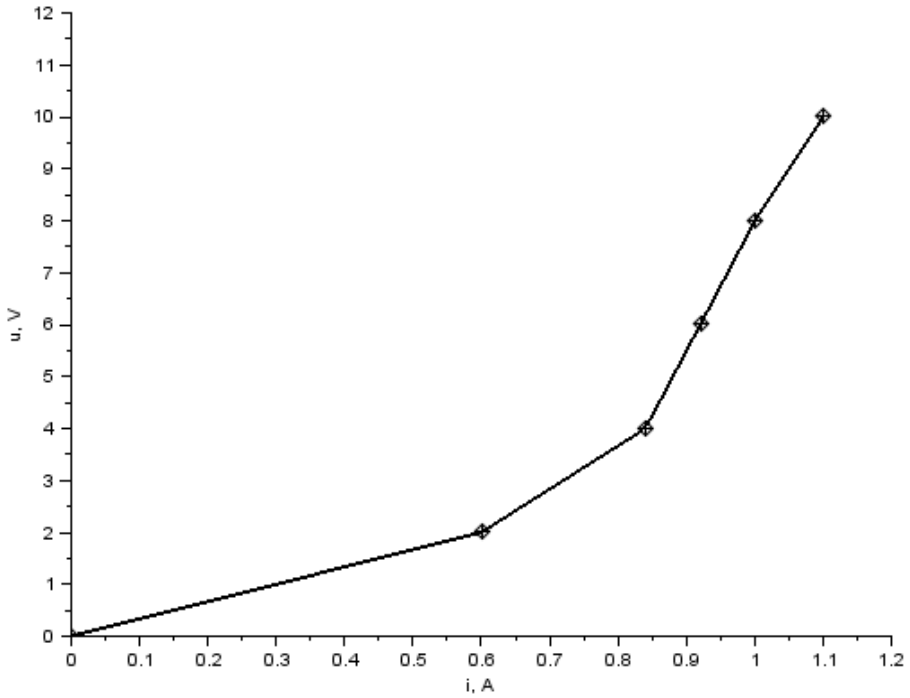


Wykres 9.1. Charakterystyka *u* = *f*(*i*) elementu elektrycznego
Źródło: opracowanie własne

Jaka funkcja matematyczna opisuje charakterystykę tego elementu? Jak zmienia się jego rezystancja dynamiczna? Jak prawidłowo wykreślić charakterystykę na podstawie punktów pomiarowych? Tego rodzaju problemy można rozwiązywać za pomocą algorytmów interpolacji i aproksymacji.

Różnicę pomiędzy zagadnieniami interpolacji i aproksymacji najlepiej widać na wykresach 9.2 i 9.3. Stojąc przed problemem narysowania wykresu przeciętny student pierwszego roku wykonana zapewne wykres 9.2. To co zaproponuje to liniowa interpolacja wyników pomiarowych – odcinki łączą się w punktach pomiarowych (węzłach interpolacji). Czy tak wygląda charakterystyka badanego elementu? Raczej nie. Gdyby użyć skomputeryzowanego stanowiska pomiarowego można by wyznaczyć pełną charakterystykę i wykonać wykres składający się wyłącznie z punktów pomiarowych. Jednak nawet takie żmudne

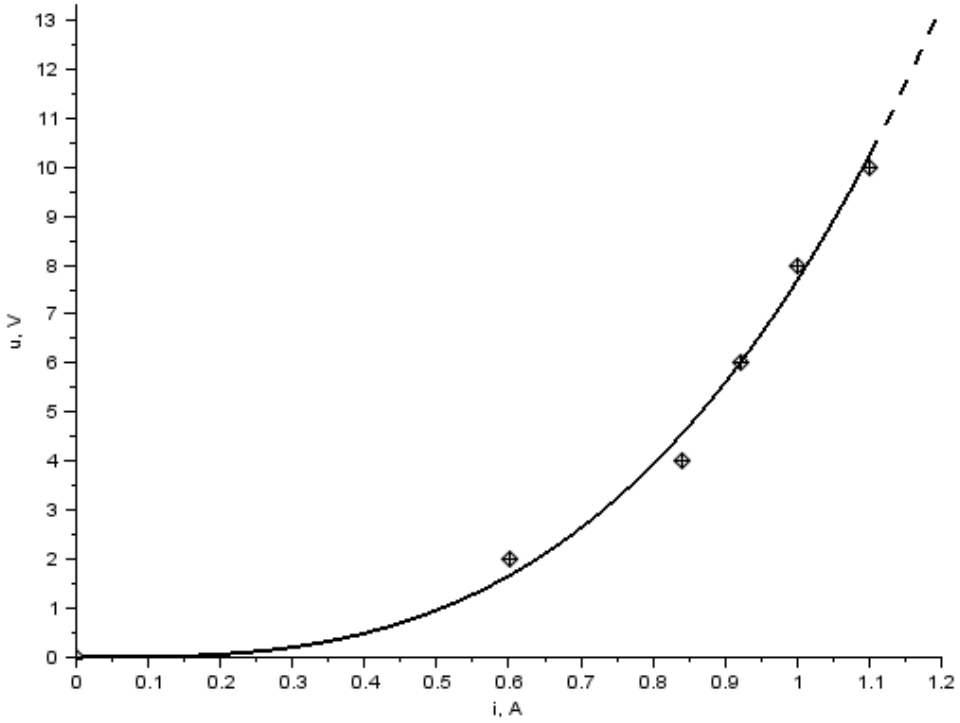
postępowanie nie dałoby odpowiedzi na temat funkcji matematycznej opisującej element elektryczny.



Wykres 9.2. Interpolacja liniowa charakterystyki *u* = *f*(*i*) elementu elektrycznego
Źródło: opracowanie własne

W rozwiązaniu tego dylematu pomoże aproksymacja. Korzystając z dostępnych źródeł wiedzy należy przyjąć znany opis matematyczny badanego zjawiska lub postawić możliwą do udowodnienia własną hipotezę na ten temat. Następnie posługując się jedną z metod minimalizacji rozbieżności pomiędzy wynikami pomiarowymi (węzłami) i funkcją aproksymującą wyznaczyć parametry funkcji aproksymującej.

Na wykresie 9.3 pokazano rezultaty takiego postępowania. Za pomocą metody najmniejszych kwadratów wyznaczono parametry a i b funkcji aproksymującej o postaci $u = ai^b$ i wykonano jej wykres linią ciągłą. Jak widać wykres nie przechodzi przez wszystkie węzły ale w ramach przyjętych założeń matematycznych optymalnie aproksymuje opisaną wynikami pomiarów zależność. W ten sposób można nie tylko wyznaczyć brakujące fragmenty charakterystyki pomiędzy węzłami, ale uzyskać także pogląd na to jak przebiega charakterystyka dla wartości znajdujących się poza zakresem zbadanym w laboratorium. Postępowanie takie nosi nazwy ekstrapolacji a jego skutki zostały oznaczone na wykresie 9.3 za pomocą linii przerywanej.



Wykres 9.3. Aproksymacja charakterystyki *u* = *f*(*i*) elementu elektrycznego
Źródło: opracowanie własne

9.1. Interpolacja liniowa

Ten rodzaj interpolacji jest chyba najbardziej znany ponieważ polega na wyznaczeniu równania prostej przechodzącej przez dwa punkty (x_i, y_i), (x_{i+1}, y_{i+1}) i wykorzystaniu go do wyznaczania wartości pośrednich y_k

$$y_k = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}\left(x_k - x_i\right).$$

Podczas kreślenia wykresu interpolacja liniowa sprowadza się do przykładania linijki i rysowania odcinków łączących dwa kolejne węzły interpolacji.

Skrypt Scilab-a pozwalający na interpolację liniową przedstawiono poniżej.

```
clear
clc
xdel(winsid())
```

```
// wyniki pomiarów u, i
u = [0 2 4 6 8 10];
i = [0 0.60 0.84 0.92 1.00 1.10];

// wykres wyników pomiarowych za pomocą symboli nr 8
plot2d(i, u, style=-8, rect=[0 0 1.2 12]);

// opis wykresu i osi
xtitle('u = f(i)', 'i, A', 'u, V')

// zbiór wartości, dla których zostanie wykonana
// interpolacja liniowa: 101 punktów
// w przedziale od 0 do 1,1 A
ip = linspace(0, 1.1, 101);

// interpolacja liniowa na podstawie węzłów
// w macierzy dwuwierszowej [i; u]
up = interpln([i; u], ip);

// wykres interpolacji linią ciągłą
plot2d(ip, up);
```

9.2. Interpolacja wielomianowa

Dla zbioru danych $\{(x_0, y_0), (x_1, y_1),\ldots (x_n, y_n)\}$ można znaleźć wielomian stopnia n

$$y = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

przechodzący przez wszystkie węzły interpolacji.

Wyznaczenie współczynników wielomianu sprowadza się do rozwiązania układu n+1 równań liniowych

$$\mathbf{XA} = \mathbf{Y} \Leftrightarrow \begin{cases} a_0 + a_1 x_0 + a_2 x_0^2 + \ldots + a_n x_0^n = y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \ldots + a_n x_1^n = y_1 \\ \vdots \\ a_0 + a_1 x_n + a_2 x_n^2 + \ldots + a_n x_n^n = y_n \end{cases}$$

gdzie niewiadomymi są $a_0, a_1, \ldots a_n$.

Skrypt Scilab-a rozwiązujący to zagadnienie może wyglądać tak:

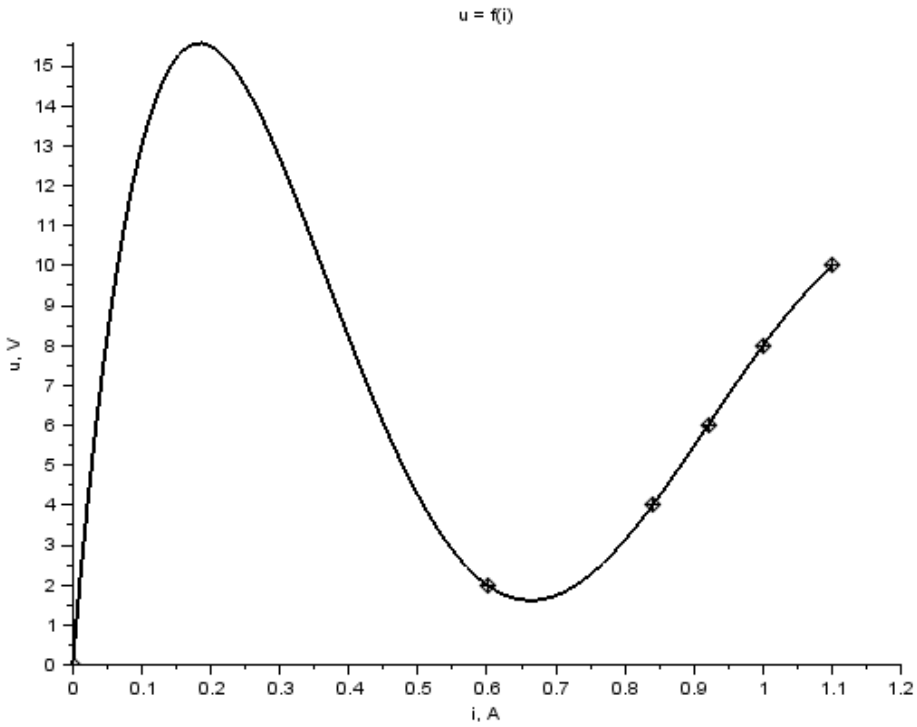
```
clear
clc
xdel(winsid())

// dane pomiarowe
napiecie = [0 2 4 6 8 10];
prad = [0 0.60 0.84 0.92 1.00 1.10];

// liczba punktów pomiarowych (kolumn macierzy napięcia)
n = size(napiecie, 'c');

// wykres danych pomiarowych z opisem
plot2d(prad, napiecie, style=-8, rect=[0 0 1.2 12]);
xtitle('u = f(i)', 'i, A', 'u, V')

// uformowanie macierzy układu równań
I = zeros(n , n)
for i = 1:n
  I(i, 1) = 1
  for j = 2:n
    I(i, j) = prad(i)^(j-1)
  end
end

// rozwiązanie układu równań i * a = napięcie
// w poszukiwaniu współczynników wielomianu
A = linsolve(I, -napiecie')
disp(A, 'Współczynniki wielomianu: ');

// wielomian na bazie uzyskanych współczynników
wielomian = poly(A, 'i', 'coeff')
disp(wielomian)

// zakres interpolacji
ip = linspace(0, 1.1, 101);

// obliczenie wartości wielomianu
// w powyższym zakresie
up = horner(wielomian, ip);

// wykres przebiegu funkcji interpolującej
plot2d(ip, up);
```

Na wykresie 9.4 pokazano wyniki uzyskane podczas interpolacji wielomianowej. Funkcja interpolująca została numerycznie określona jako

$$u = 204{,}384i - 869{,}354i^2 + 1340{,}537i^3 - 874{,}014i^4 + 206{,}447i^5.$$

Matematyczne zasady interpolacji wielomianowej zostały spełnione ale czy tak wygląda charakterystyka badanego elementu? Raczej nie.

Interpolacja wielomianowa ma jeszcze jedną wadę. Jeżeli zbiór danych jest duży należy rozwiązać równie duży układ równań, a wielomian interpolacyjny jest bardzo wysokiego stopnia. W takiej sytuacji mogą zacząć pojawiać się problemy związane z dokładnością obliczeń i dokładnością reprezentacji wartości liczbowych.



Wykres 9.4. Interpolacja wielomianowa danych pomiarowych
Źródło: opracowanie własne

9.3. Interpolacja funkcją sklejaną

Interpolacja funkcją sklejaną to metoda interpolacji polegająca na użyciu szeregu wielomianów niskiego stopnia o tak dobranych współczynnikach, aby funkcja interpolująca była ciągła i miała w węzłach interpolacji ciągłą pierwszą i często także wyższe pochodne.

Najczęściej wykorzystywanym do sklejania wielomianem jest wielomian trzeciego stopnia:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 .$$

W takiej sytuacji jest to sześcienna funkcja sklejana. Funkcja musi być ciągła. Jej pierwsza pochodna (nachylenie):

$$y' = a_1 + 2a_2x + 3a_3x^2$$

oraz druga pochodna (krzywizna):

$$y'' = 2a_2 + 6a_3x,$$

także muszą spełniać warunek ciągłości we wszystkich wewnętrznych węzłach interpolacji.

Przy wykorzystaniu sześciennej funkcji sklejanej dla interpolacji zbioru węzłów o liczebności n należy znaleźć $4(n–1) = 4n–4$ wartości niewiadomych. Biorąc pod uwagę, że funkcja sklejana ma być ciągła tzn. w każdy węźle wartości sąsiednich wielomianów są takie same i równe wartości funkcji interpolowanej, można zapisać $2(n–1)$ równań. Warunek ciągłości nachylenia (pierwszej pochodnej) we wszystkich wewnętrznych węzłach pozwala zapisać kolejne $(n–2)$ równań. Podobnie warunek ciągłości krzywizny (drugiej pochodnej) pozwala na utworzenie $(n–2)$ równań. Łącznie daje to $4n–6$ równań. Brakujące dwa równania można zapisać:

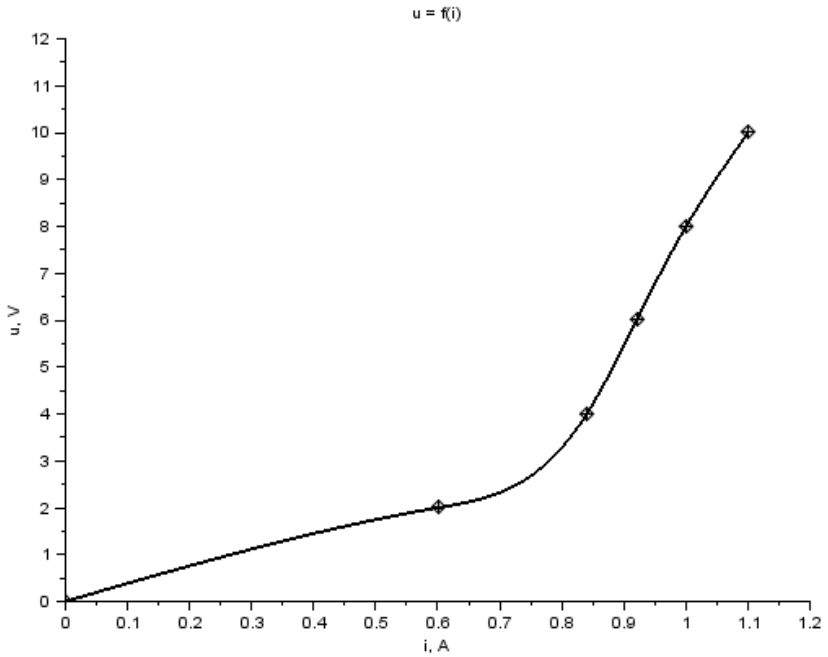
- wybierając dowolne nachylenie (wartość pierwszej pochodnej) w węzłach skrajnych,
- wybierając zerową krzywiznę (wartość drugiej pochodnej) w węzłach skrajnych,
- wybierając krzywiznę w węzłach skrajnych taką jak w węzłach sąsiednich,
- stosując liniową ekstrapolację krzywizny dla węzłów (x_2, y_2) i (x_3, y_3) oraz (x_{n-2}, y_{n-2}) i (x_{n-1}, y_{n-1}) do wyznaczenia krzywizny w węzłach skrajnych,
- stosując warunek ciągłości trzeciej pochodnej $y''' = 6a_3$ w węzłach (x_2, y_2) oraz (x_{n-1}, y_{n-1}).

Skrypt Scilab-a rozwiązujący zagadnienie interpolacji sześcienną funkcją sklejaną przedstawiono poniżej.

```
clear
clc
xdel(winsid())

// dane pomiarowe
napiecie = [0 2 4 6 8 10];
prad = [0 0.60 0.84 0.92 1.00 1.10];

// wykres danych pomiarowych z opisem
plot2d(prad, napiecie, style=-8, rect=[0 0 1.2 12]);
xtitle('u = f(i)', 'i, A', 'u, V')

// interpolacja danych sześcienną funkcją sklejaną,
```

```
// wyznaczenie pochodnych funkcji sklejanej,
// 'natural' - zerowa krzywizna na skrajach
pochodne = splin(prad, napiecie, 'natural');

// zakres interpolacji
ip = linspace(0, 1.1, 101);

// wyznaczenie wartości funkcji interpolującej
// na podstawie pochodnych funkcji sklejanej
// up - wartości funkcji (napięcia)
// up1, up2, up3 - pierwsza, druga, trzecia pochodna
[up up1 up2 up3] = interp(ip, prad, napiecie, pochodne)

// wykres przebiegu funkcji interpolującej
plot2d(ip, up);
```

Przy wykorzystaniu aproksymacji funkcją sklejaną o zerowej krzywiźnie na skrajach przedziału (oznaczenie ‘natural’) uzyskano wykres 9.5.



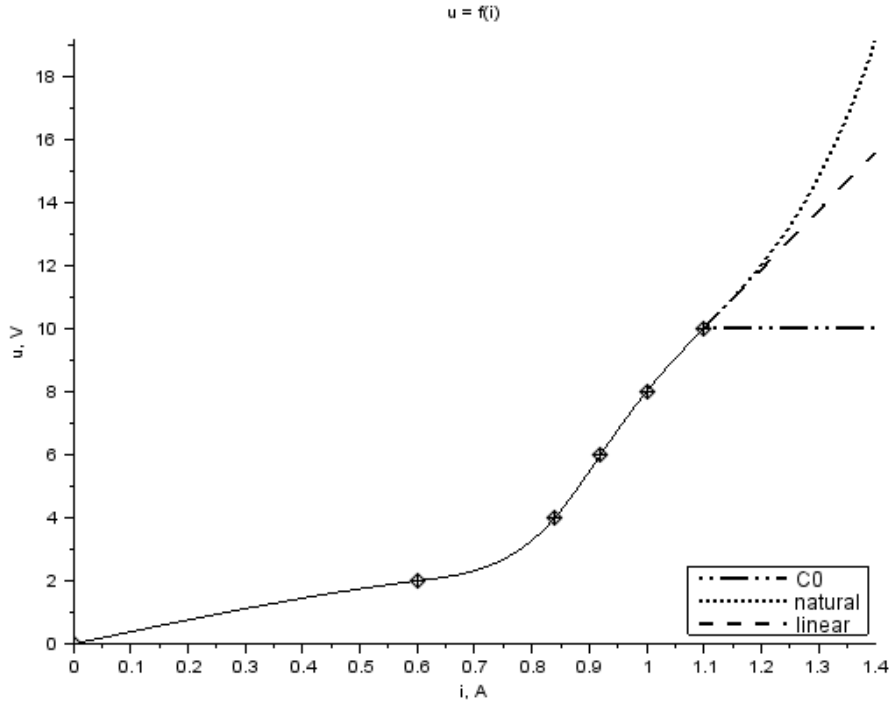
Wykres 9.5. Interpolacja sześcienną funkcją sklejaną („natural”)
Źródło: opracowanie własne

Rozszerzając w powyższym skrypcie zakres interpolacji oraz modyfikując wywołanie podprogramu **interp**() odpowiedzialnego za obliczanie wartości

funkcji interpolującej można uzyskać ekstrapolację charakterystyki. Zmiany kodu zostały uwidocznione poniżej.

```
// zakres interpolacji oraz ekstrapolacji do 1,4 A
ip = linspace(0, 1.4, 101);


// wyznaczenie wartości funkcji interpolującej
// z ekstrapolacją typu C0
[up up1 up2 up3] = interp(ip, prad, napiecie, pochodne, 'C0')
```

Istnieje kilka metod ekstrapolacji dostępnych przy użyciu funkcji **interp**(). Trzy najbardziej charakterystyczne pokazano na wykresie 9.6. Typ 'C0' ekstrapoluje charakterystykę za pomocą wartości stałej pobranej ze skrajnego węzła. Ekstrapolacja 'natural' jest obliczana według funkcji trzeciego stopnia, która interpolowała wartości pomiędzy przedostatnim i skrajnym węzłem. Typ 'linear' pozwala na ekstrapolację funkcją liniową o nachyleniu takim jakie miała funkcja interpolująca w skrajnym węźle.



Wykres 9.6. Interpolacja funkcją sklejaną z ekstrapolacją od 1,1 do 1,4 A
Źródło: opracowanie własne

9.3. Aproksymacja

Celem aproksymacji jest dobranie funkcji aproksymującej, która zgodnie z przyjętym kryterium będzie najdokładniej odtwarzała przebieg zależności aproksymowanej.

Istnieją dwa główne zagadnienia aproksymacji. W pierwszym przypadku wartości funkcji aproksymowanej znane są jedynie jako współrzędne punktów (węzłów aproksymacji) wyznaczonych na drodze obserwacji i to zazwyczaj obarczonej błędami pomiarowymi. W drugim przypadku poszukiwana jest przybliżona forma znanej, ale zbyt skomplikowanej do dalszej analizy funkcji matematycznej.

Kryteria jakości aproksymacji mogą być formułowane na podstawie wartości odchyłek między wartością funkcji aproksymowanej i aproksymującej jako:

- najmniejsza wartość bezwzględna odchyłki maksymalnej (aproksymacja jednostajna),
- najmniejsza suma wartości bezwzględnych odchyłek,
- najmniejsza suma kwadratów odchyłek (aproksymacja średniokwadratowa).

Tylko w najprostszych przypadkach takich jak aproksymacja liniowa lub aproksymacja wielomianem uogólnionym obliczenia są stosunkowo proste do realizacji. W trudniejszych sytuacjach z pomocą przychodzą algorytmy optymalizacyjne.

Scilab został wyposażony w uniwersalną funkcję **datafit**(), która służy do obliczeń aproksymacyjnych dla zadanego zbioru wyników pomiarowych (węzłów). Funkcja pozwala na wyznaczenie wartości współczynników poszukiwanej funkcji pod kątem minimalizacji sumy kwadratów odchyłek. Należy pamiętać, że w przypadku dużych zbiorów węzłów aproksymacji obliczenia optymalizacyjne mogą potrwać stosunkowo długo (nawet kilkanaście minut). Na ich przebieg mają wpływ zadane przez użytkownika wartości początkowe parametrów funkcji aproksymującej i może zdarzyć się, że w przypadku problemów drobna modyfikacja pozwoli na uzyskanie poprawnego wyniku.

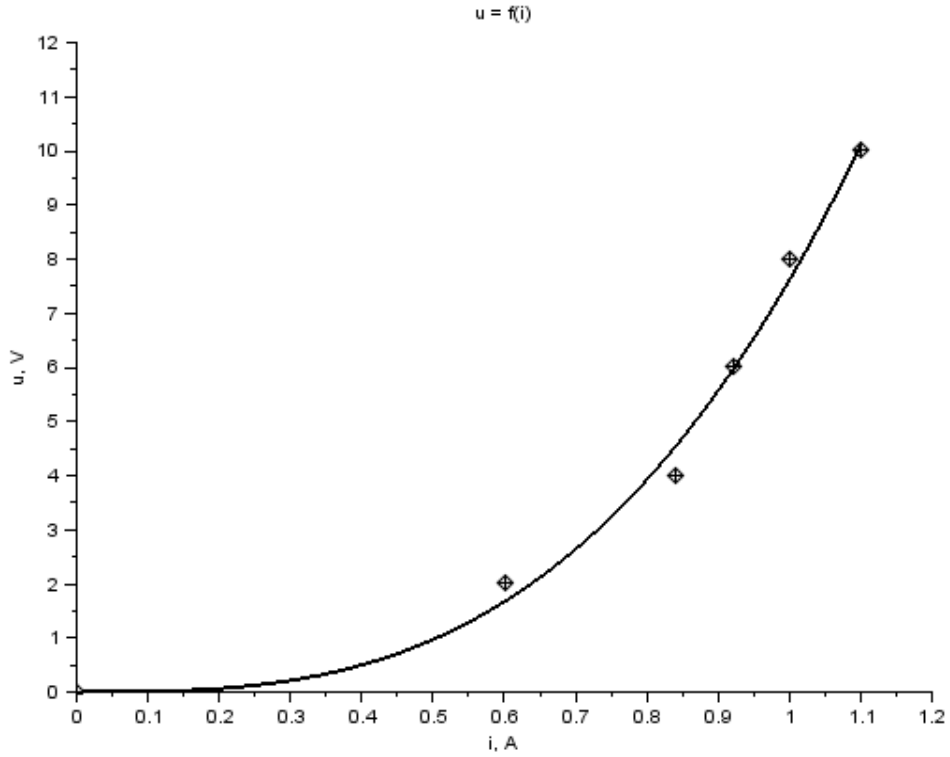
Przykład skryptu obliczeniowego do aproksymacji wartości pomiarów za pomocą funkcji potęgowej został przedstawiony poniżej.

```
clear
clc
// czyszczenie okna graficznego
xdel(winsid())

// dane pomiarowe
napiecie = [0 2 4 6 8 10];
prad = [0 0.60 0.84 0.92 1.00 1.10];
```

```
// macierz powyższych wyników pomiarowych
daneXY = [prad; napiecie];

// wykres danych pomiarowych z opisem
plot2d(prad, napiecie, style=-8, rect=[0 0 1.2 12]);
xtitle('u = f(i)', 'i, A', 'u, V')

// funkcja potęgowa aproksymująca wyniki pomiarów
// postaci y = a * x^b
// gdzie a, b to parametry funkcji
function y = fa(x, p)
  y = p(1) * x^p(2);
endfunction

// funkcja oceny błędu aproksymacji
function blad = fkryt(parametr, dane)
  x = dane(1);
  y = dane(2);
  blad = y - fa(x, parametr);
endfunction

// wartości początkowe parametrów aproksymacji
p0 = [1; 1];

// obliczenie parametrów aproksymacji
// funkcją datafit()
[parametry, blad] = datafit(fkryt, daneXY, p0);

// wykres wyników aproksymacji
ip = linspace(0, 1.1, 101);
plot2d(ip, fa(ip, parametry));

disp(blad, 'wartość błędu aproksymacji: ' );
disp(parametry, 'parametry aproksymacji: ');
```

Wynikiem działania tego przykładu są: wartość błędu aproksymacji, optymalne wartości parametrów funkcji aproksymującej, wykresy danych pomiarowych i przebiegu funkcji aproksymującej (wykres 9.7).

```
  wartość błędu aproksymacji:
      0.5594433

   parametry aproksymacji:
      7.6250776
      2.9744546
```



Wykres 9.7. Aproksymacja danych pomiarowych funkcją potęgową
Źródło: opracowanie własne

Zmieniając fragment kodu dotyczący funkcji aproksymującej

```
// funkcja aproksymująca wyniki pomiarów
// postaci y = a*x*x + b*x + c
function y = fa(x, p)
    y = p(1)*x^2 + p(2)*x + p(3)
endfunction
```

oraz wartości początkowe parametrów

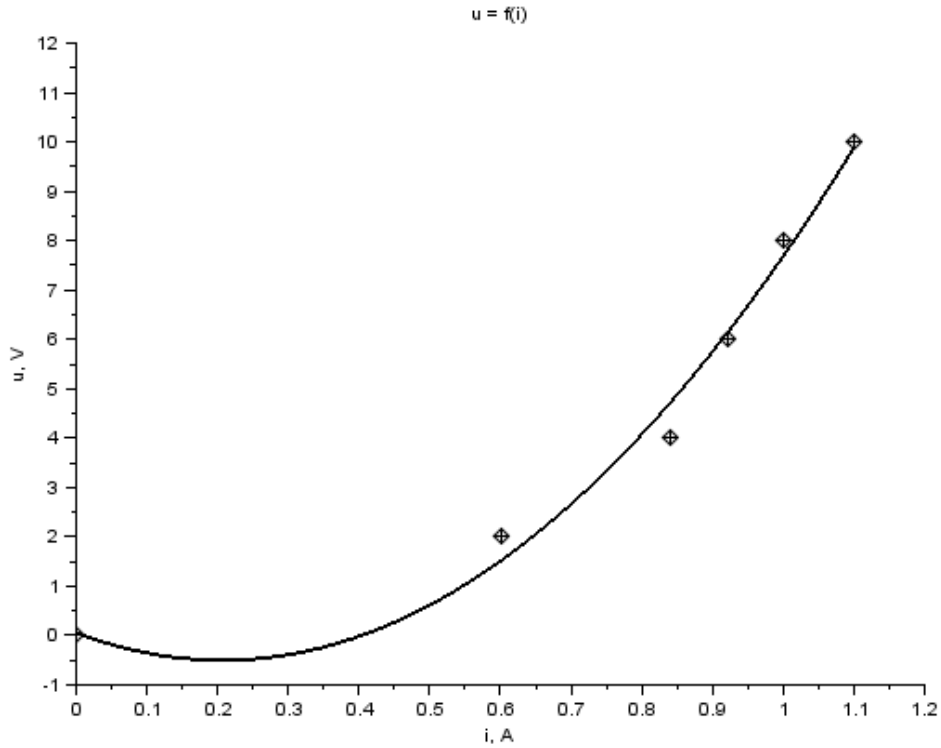
```
// wartości początkowe parametrów aproksymacji
p0 = [1; 1; 1];
```

uzyskano następujące wyniki dla aproksymacji za pomocą funkcji kwadratowej:

```
wartość błędu aproksymacji:
    0.8818964
```

```
parametry aproksymacji:
   13.044685
 - 5.4107265
   0.0602749
```

Wykres 9.8 należy oceniać nie tylko na podstawie błędu aproksymacji, który w porównaniu z poprzednim przypadkiem posiada tylko nieco większą wartość, ale przede wszystkim na podstawie wiedzy o badanym zjawisku. Należy zauważyć, że element pasywny, który był tutaj przykładem nie może mieć fragmentu charakterystyki $u = f(i)$ o wartościach ujemnych dla dodatnich odciętych. Aproksymacja funkcją kwadratową mimo, że numerycznie poprawna nie jest odpowiednia dla rozpatrywanego przypadku elementu elektrycznego.



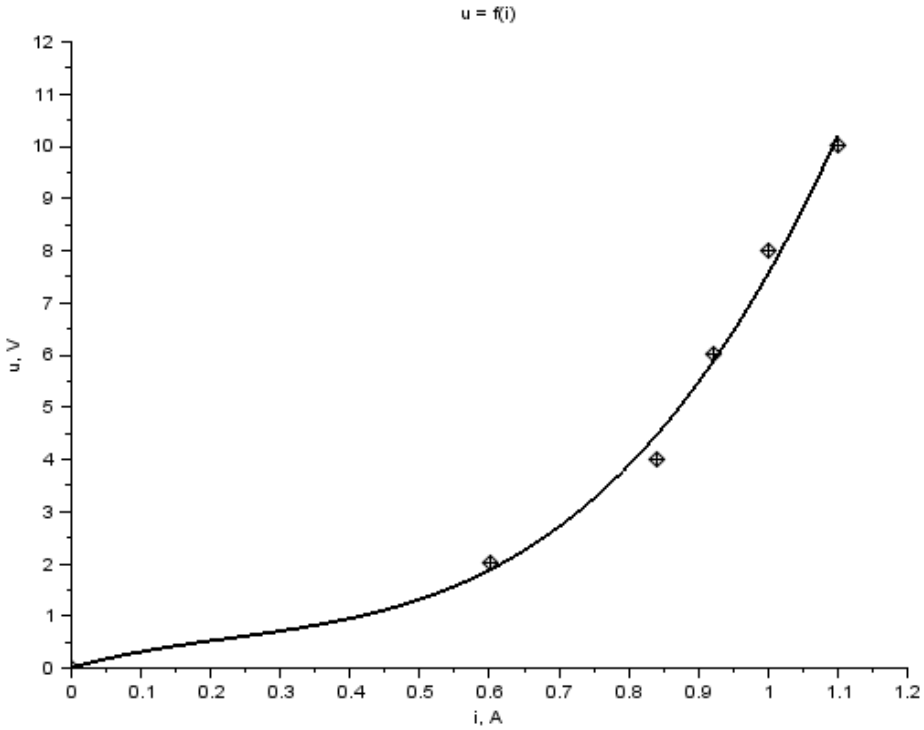
Wykres 9.8. Aproksymacja danych pomiarowych funkcją kwadratową
Źródło: opracowanie własne

Dużo lepsze pod względem dokładności i fizycznie uzasadnione dopasowanie można uzyskać dla funkcji sześciennej:

```
// funkcja aproksymująca wyniki pomiarów
// postaci y = a*x^3 + b*x^2 + c*x + d
function y = fa(x, p)
    y = p(1)*x^3 + p(2)*x^2 + p(3)*x + p(4)
endfunction
```

Oczywiście w skrypcie zmieniono także liczbę warunków początkowych. Uzyskane wyniki:

```
wartość błędu aproksymacji:
   0.4914074

parametry aproksymacji:
   12.192011
 - 8.4016849
   3.7716837
   0.0061595
```



Wykres 9.9. Aproksymacja danych pomiarowych funkcją sześcienną
Źródło: opracowanie własne

Zadanie

Dla podanego zestawu wyników pomiarowych należy wykonać:

a) interpolację liniową,
b) interpolację wielomianową,
c) interpolację sześcienną funkcją sklejaną,
d) aproksymacje wybranymi funkcjami w celu zminimalizowania błędu aproksymacji,
e) ekstrapolacje wyników w zadanym przedziale z użyciem funkcji sklejanej wyznaczonej podczas interpolacji oraz funkcji aproksymującej uznanej za najlepszą.

Sprawozdanie

Do wydruku sprawozdania należy dołączyć nośnik zawierający skrypt obliczeniowy. Sprawozdanie powinno zawierać:

a) opis zadania z danymi przyznanymi grupie,
b) treść skryptu Scilab-a rozwiązującego zadania (z komentarzami programisty),
c) opisane wykresy, równania funkcji aproksymujących z wartościami współczynników,
d) zestawienie porównawcze błędów aproksymacji dla różnych funkcji,
e) wnioski dotyczące wyników.

10. Równania różniczkowe

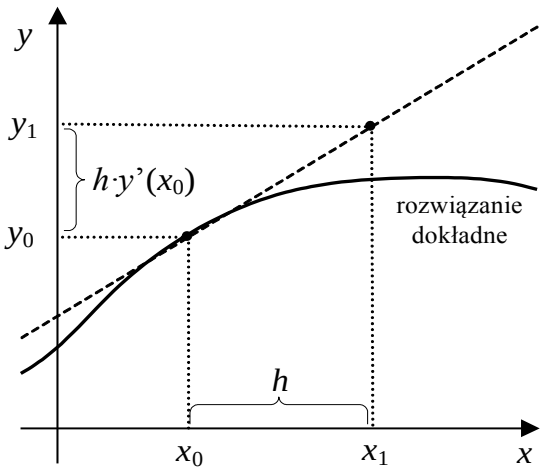
Zjawiska dynamiczne zachodzące w obwodach elektrycznych opisywane są językiem matematycznym za pomocą równań różniczkowych. Rozwiązanie ich analitycznie jest często procesem trudnym i czasochłonnym. W celu zautomatyzowania procesu rozwiązywania równań różniczkowych opracowana kilka metod umożliwiających uzyskanie wyników przybliżonych. W środowiskach obliczeniowych, takich jak Scilab, zaimplementowane są funkcje realizujące te zadanie.

10.1. Metody jednokrokowe

Algorytmy jednokrokowe należą do metod zgrubnych. Charakteryzują się dobrą dokładnością pod warunkiem przyjęcia niewielkiej wartości przyrostu w pojedynczym kroku. Dokładność prostych algorytmów całkowania numerycznego można poprawiać stosując bardziej złożone sposoby wyznaczania wartości funkcji $f(x, y)$.

10.1.1. Metoda Eulera

Metoda Eulera należy do algorytmów jawnych. Obliczenia odbywają się krokowo zaczynając od wartości początkowych (x_0, y_0). W każdej iteracji obliczane są następne wartości (x_{n+1}, y_{n+1}).



Wykres 10.1. Pojedynczy krok metody Eulera
Źródło: opracowanie własne

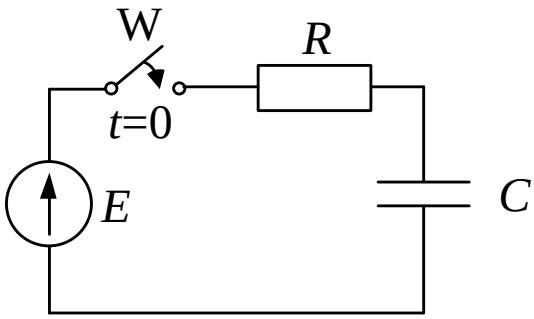
Przyjmując stały krok metody h można obliczyć:

- kolejną wartość zmiennej niezależnej $x_{n+1} = x_n + h$,

- wartość pochodnej $y' = f(x, y) = \frac{\Delta y}{h}$,
- oraz kolejną wartość zmiennej zależnej
  $y_{n+1} = y_n + \Delta y = y_n + hy' = y_n + h \cdot f(x_n, y_n)$.

Metoda Eulera nazywana jest czasami błędnie metodą Rungego-Kutty rzędu pierwszego.

W celu zobrazowania metody obliczeniowej zostanie rozpatrzone następujące zadanie: wyznaczyć przebieg napięcia na kondensatorze w stanie nieustalonym w obwodzie szeregowym RC przy wymuszeniu napięciem stałym (rysunek 10.1) i zerowym warunku początkowym. Dane są następujące wartości elementów: E = 10 V, R = 5 kΩ, C = 100 μF.



Rysunek 10.1. Schemat obwodu RC zasilanego ze źródła napięcia stałego do analizy stanu nieustalonego
Źródło: opracowanie własne

Równanie II prawa Kirchhoffa dla tego obwodu

$$E - u_R - u_C = 0,$$

$$E - Ri - u_C - 0,$$

$$E - RC\frac{\mathrm{d}u_C}{\mathrm{d}t} - u_C = 0$$

$$\frac{\mathrm{d}u_C}{\mathrm{d}t} = \frac{-u_C + E}{RC}$$

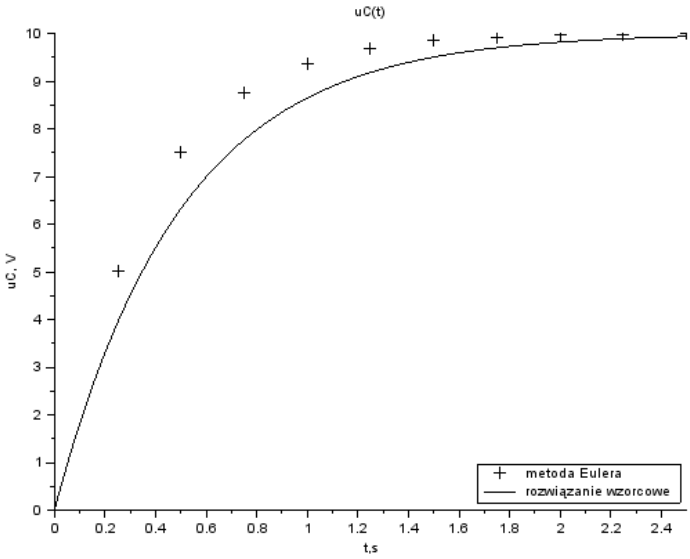
Fakt, że rozwiązanie tego równania różniczkowego jest znane z wykładów z elektrotechniki

$$u_C = E\left(1 - e^{-\frac{t}{RC}}\right)$$

daje możliwość porównania wyników uzyskanych metodą Eulera.
Przykładowy skrypt Scilab-a:

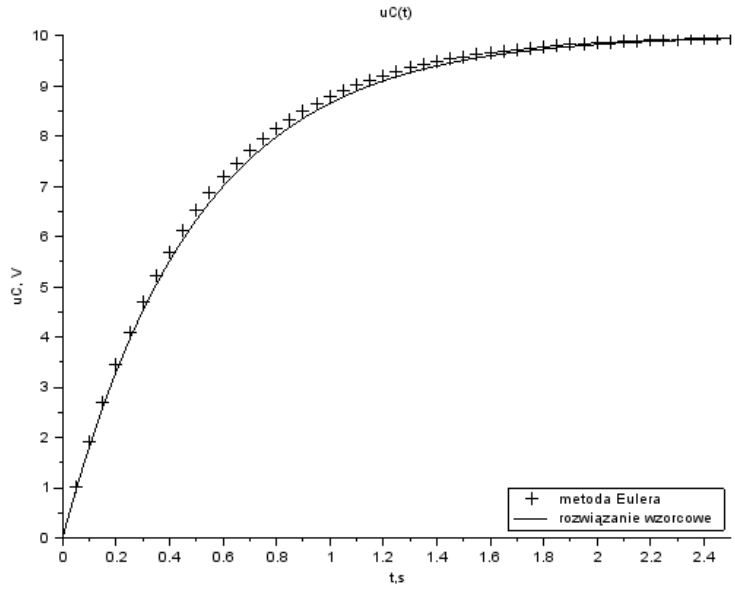
```
clear;
clc;
xdel(winsid());
// parametry zadania
E = 10;
R = 5e3;
C = 100e-6;
// stała czasu obwodu RC
tau = R*C;
// definicja funkcji y' = f(t, y) dla obwodu RC
// ładowanego ze źródła napięcia stałego
function w=rozn(t, u)
  w = (-u + E)/tau;
endfunction
// funkcja wzorcowa opisująca przebieg napięcia
// na kondensatorze wyznaczona metodą klasyczną
function w=wzorcowa(t)
  w = E*(1 - exp(-t/tau));
endfunction
// funkcja rozwiązująca zadanie metodą Eulera
// parametry: czas początkowy, czas końcowy,
// przyrost czasu, warunek początkowy dla y
function [t, y]=Euler(t0, tk, h, y0)
  N = (tk - t0)/h;      // liczba kroków
  t(1) = t0;            // inicjalizacja wektora czasu
  y(1) = y0;            // inicjalizacja wektora rozwiązań
  for n = 1:N
    t(n+1) = t(n) + h;
    y(n+1) = y(n) + h * rozn(t(n), y(n));
  end
endfunction
// obliczenie rozwiązania metodą Eulera
// czyli napięcia na kondensatorze
// od 0 to 5*tau, krok tau/2 czyli 10 kroków,
// zerowy warunek początkowy
[eu_t, eu_uc] = Euler(0, 5*tau, tau / 2, 0);
// wykres rozwiązania Eulera z użyciem symboli nr 1
plot2d(eu_t, eu_uc, style = -1);
// opisy: wykresu, osi x, osi y
```

```
xtitle('uC(t)', 't,s', 'uC, V');
// wykres rozwiązania dokładnego z użyciem linii ciągłej
t = linspace(0, 5 * tau, 201);
plot2d(t, wzorcowa(t), rect = [0, 0, 5 * tau, E]);
// legenda do wykresu
legends(['metoda Eulera', 'rozwiązanie wzorcowe'],
        [-1,1], opt='lr')
// obliczenie napięcia na rezystorze
ur = E - eu_uc;
// i prądu w obwodzie
i = ur/R;
// otwarcie nowego okna graficznego
scf;
// górny wykres w oknie
subplot(211)
// to wykres napięcia na rezystorze
plot2d(eu_t, ur);
// z opisem
xtitle('uR(t)', 't, s', 'uR, V');
// dolny wykres w oknie
subplot(212)
// to wykres prądu
plot2d(eu_t, i);
// z opisem
xtitle('i(t)', 't,s', 'i, A');
```



Wykres 10.2. Rozwiązanie zadania metodą Eulera: napięcie na kondensatorze (krok 0,5τ)
Źródło: opracowanie własne

Ponieważ rozwiązanie jest bardzo niedokładne (wykres 10.2) zmieniono liczbę kroków algorytmu Eulera z 10 do 50 i powtórzono obliczenia (wykres 10.3).



Wykres 10.3. Rozwiązanie zadania metodą Eulera: napięcie na kondensatorze (krok 0,1τ)
Źródło: opracowanie własne

10.1.2. Metoda Rungego-Kutty

Nawet tylko na podstawie wykresów 10.2 i 10.3 można zauważyć jak bardzo wrażliwa jest metoda Eulera na długość kroku rozwiązania *h*. Oczywiście zmniejszanie przyrostu *h* poprawia dokładność ale znacząco spowalnia obliczenia.

Można wybrać inną drogę poprawy dokładności. Ulepszenie zaproponowane przez niemieckich matematyków Carla Rungego i Martina Wilhelma Kuttę polega na obliczaniu wartości pochodnej w połowie przedziału:

- pochodna na początku rozpatrywanego przedziału $k_1 = f(x_n, y_n)$,
- oszacowanie pochodnej w środku przedziału za pomocą wartości pochodnej na początku przedziału $k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot k_1)$,
- kolejna wartość zmiennej zależnej obliczana na podstawie wartości pochodnej z połowy przedziału $y_{n+1} = y_n + \Delta y = y_n + h \cdot y' = y_n + h \cdot k_2$.

Powyższe postępowanie nosi nazwę algorytmu Rungego-Kutty rzędu drugiego (RK2).

Usprawnienie obliczeń może pójść znacznie dalej. Kolejny pomysł polegał na zastosowaniu bardziej złożonego sposobu wyznaczania wartości pochodnej – jako średniej ważonej czterech jej oszacowań:

- k_1, k_2 – obliczane jak w algorytmie RK2,
- drugie oszacowanie pochodnej w połowie przedziału

$$k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot k_2),$$

- oszacowanie pochodnej na końcu przedziału $k_4 = f(x_n + h, y_n + h \cdot k_3)$
- kolejna wartość zmiennej zależnej obliczana na podstawie średniej ważonej zgromadzonych wartości pochodnej funkcji

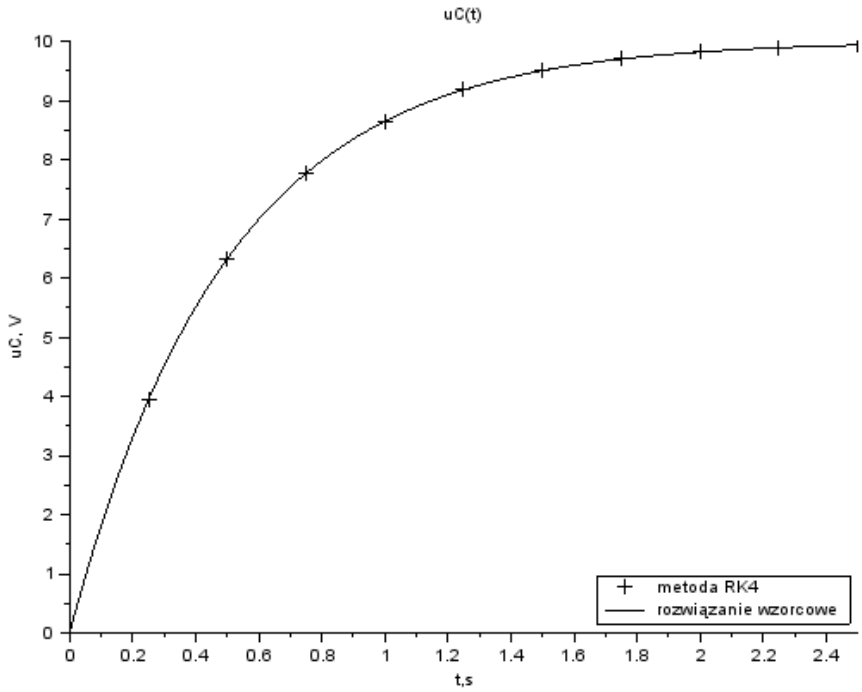
$$y_{n+1} = y_n + \Delta y = y_n + h \cdot y' = y_n + \frac{h}{6} \cdot (k_1 + 2k_2 + 2k_3 + k_4).$$

Przedstawione postępowanie nosi nazwę algorytmu Rungego-Kutty rzędu czwartego (RK4) i mimo tego, że jest chyba jedną z najpopularniejszych metodą analizy równań różniczkowych to nie wyczerpuje tego zagadnienia.

Przykładowy skrypt rozwiązujący poprzednie zadanie metodą RK4:

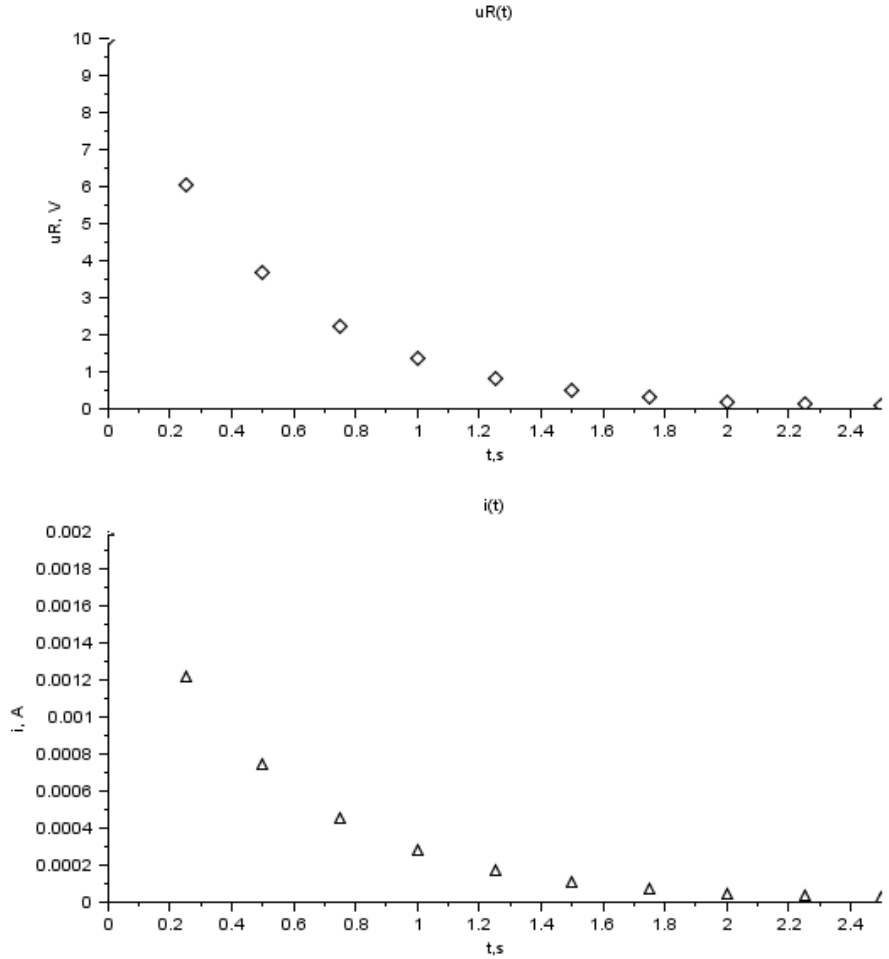
```
clear;
clc;
xdel(winsid());

// parametry zadania
E = 10;
R = 5e3;
C = 100e-6;
// stała czasu obwodu RC
tau = R*C;
// definicja funkcji y' = f(t, y) dla obwodu RC
// zasilanego ze źródła napięcia stałego
function w=rozn(t, u)
  w = (-u + E)/tau;
endfunction
// funkcja wzorcowa opisująca przebieg napięcia
// na kondensatorze wyznaczona metodą klasyczną
function w=wzorcowa(t)
  w = E*(1 - exp(-t/tau));
endfunction
// rozwiązanie metodą Runge-Kutta 4
// parametry: czas początkowy, czas końcowy,
// przyrost czasu, warunek początkowy dla y
function [t, y]=RK4(t0, tk, h, y0)
```

```
  N = (tk - t0)/h;
  t(1) = t0;
  y(1) = y0;
  for n = 1:N
    t(n+1) = t(n) + h;
    k1 = rozn(t(n), y(n));
    k2 = rozn(t(n) + h/2, y(n)+ h/2 * k1);
    k3 = rozn(t(n) + h/2, y(n)+ h/2 * k2);
    k4 = rozn(t(n) + h  , y(n)+ h   * k3);
    y(n+1) = y(n) + h/6 * (k1 + 2*k2 + 2*k3 + k4);
  end
endfunction
// obliczenie rozwiązania metodą Eulera
// od 0 to 5*tau, krok tau/2 czyli 10 kroków,
// warunek początkowy zerowy
[rk_t, rk_uc] = RK4(0, 5 * tau, tau / 2, 0);
// wykres rozwiązania RK4 z użyciem symboli nr 1
plot2d(rk_t, rk_uc, style = -1);
// nazwa wykresu i osi
xtitle('uC(t)','t, s','uC, V');
// wykres rozwiązania dokładnego z użyciem linii ciągłej
t = linspace(0, 5 * tau, 201);
plot2d(t, wzorcowa(t), rect = [0, 0, 5 * tau, E]);
//legenda do wykresu
legends(['metoda RK4', 'rozwiązanie wzorcowe'], [-1, 1],
        opt = 'lr')
// obliczenie napięcia na rezystorze
ur = E - rk_uc;
// i prądu w obwodzie
i = ur/R;
// otwarcie nowego okna graficznego
scf;
// górny wykres w oknie
subplot(211)
// to wykres napięcia na rezystorze
plot2d(rk_t, ur, style = -5);
// z opisem
xtitle('uR(t)', 't,s', 'uR, V');
// dolny wykres w oknie
subplot(212)
// to wykres prądu
plot2d(rk_t, i, style = -6);
// z opisem
xtitle('i(t)', 't,s', 'i, A');
```



Wykres 10.4. Rozwiązanie zadania metodą RK4: napięcie na kondensatorze (krok 0,5τ)
Źródło: opracowanie własne

Rozwiązanie równania różniczkowego metodą RK4 może być uznane za satysfakcjonujące więc wykreślenie pozostałych przebiegów wielkości elektrycznych ma sens (wykres 10.5).



Wykres 10.5. Rozwiązanie zadania metodą RK4 (krok 0,1τ): napięcie na rezystorze (górny wykres), prąd w obwodzie (dolny wykres)
Źródło: opracowanie własne

Zadania

1. Wskazanymi przez prowadzącego zajęcia metodami należy wyznaczyć przebiegi elektryczne w szeregowych obwodach RL i RC o zadanych parametrach w stanie nieustalonym przy wymuszeniu:
   a) stałym,
   b) sinusoidalnym.

   Przedstawić porównanie z rozwiązaniami dokładnymi.
2. Należy wyznaczyć przebiegi elektryczne w obwodzie szeregowym RLC o zadanych parametrach w stanie nieustalonym przy wymuszeniu:

a) stałym,
b) sinusoidalnym.

Zadanie rozwiązać dla następujących przypadków:

a) $R << Rkr$ – oscylacyjny, tłumiony (np. $R = R_{kr}/5$),
b) $R = R_{kr}$ – krytyczny,
c) $R >> R_{kr}$ – aperiodyczny (np. $R = 5 \cdot R_{kr}$).

gdzie $R_{kr} = 2\sqrt{\frac{L}{C}}$. Przedstawić przebiegi napięcia na kondensatorze i prądu w obwodzie (dla każdego przypadku porównanie wyników obu metod przy tej samej liczbie kroków). Zakres czasu rozwiązania dobrać stosownie do analizowanego przypadku (ustalenie wartości).

Wskazówki

Omówione wcześniej postaci metody Eulera i RK pozwalają na rozwiązywanie równań różniczkowych pierwszego rzędu. Aby rozwiązać równanie różniczkowe rzędu np. drugiego (obwód RLC) trzeba je najpierw przedstawić w postaci układu dwóch równań rzędu pierwszego, a następnie równocześnie rozwiązywać oba równania.

W powyższym zadaniu należy więc przystosować metody obliczeniowe do równoczesnego obliczania wartości dwóch funkcji. Przykład modyfikacji:

```
function [t, y1, y2] = Euler-2(t0, tk, h, y10, y20)
  N = (tk - t0)/h;
  t(1) = t0;
  y1(1) = y10;
  y2(1) = y20;
  for n = 1:N
    t(n+1) = t(n) + h;
    // y1(t) to np. prąd w obwodzie (prąd cewki)
    y1(n+1) = y1(n) + h * f1(t(n), y1(n), y2(n));
    // y2(t) to np. napięcie na kondensatorze
    y2(n+1) = y2(n) + h * f2(t(n), y1(n), y2(n));
  end
endfunction
```

Oczywiście wymagane jest wcześniejsze zdefiniowanie funkcji *f1*(*t*, *y1*, *y2*) i *f2*(*t*, *y1*, *y2*) na podstawie wyprowadzonych równań różniczkowych.

Sprawozdanie

Do wydruku sprawozdania należy dołączyć nośnik zawierający skrypt obliczeniowy. Sprawozdanie powinno zawierać:

1. opis zadań z parametrami przyznanymi grupie,
2. pełne wyprowadzenia równań różniczkowych dla podanych obwodów,
3. treść skryptu Scilab-a rozwiązującego zadania (z komentarzami programisty),
4. porównanie rozwiązania dokładnego z wynikami uzyskanymi metodami numerycznymi przedstawione na wykresach (opisanych, we właściwej skali), w razie konieczności z uwypukleniem (powiększeniem) rozbieżności pomiędzy metodami numerycznymi,
5. wnioski dotyczące wyników z oceną wpływu typu zmienności funkcji i wielkości kroku obliczeń na zbieżność i dokładność rozwiązań.

10.2. Równania I i II stopnia – analiza stanów nieustalonych metodą zmiennych stanu

Środowiska obliczeniowe takie jak Matlab i Scilab oferują narzędzia umożliwiające rozwiązywanie równań różniczkowych, za pomocą którego opisuje się językiem matematycznym zjawiska dynamiczne – stany nieustalone w obwodach elektrycznych [3]. W Scilab'ie zadanie to realizuje funkcja `ode`. Uogólnioną składnię, zawierającą najważniejsze parametry, funkcji `ode` zapisano poniżej [1].

```
y= ode([type,] y0, t0, t, f)
```

gdzie:

`y` – wyjściowy wektor lub macierz rozwiązania,

`type` – nazwa „silnika" metody rozwiązywania równań różniczkowych ("adams", "stiff", "rk", "rkf", "fix", "discrete", "roots"), wybrane metody zostały omówione we wcześniejszym ćwiczeniu,

`y0` – wektor lub macierz warunków początkowych zadania,

`t0` – wartość początkowa zmiennej względem której rozwiązywane jest zadanie,

`t` – wektor wartości zmiennej względem której rozwiązywane jest zadanie,

`f` – nazwa funkcji zawierającej definicję równań różniczkowych rozwiązywanych funkcją `ode`.

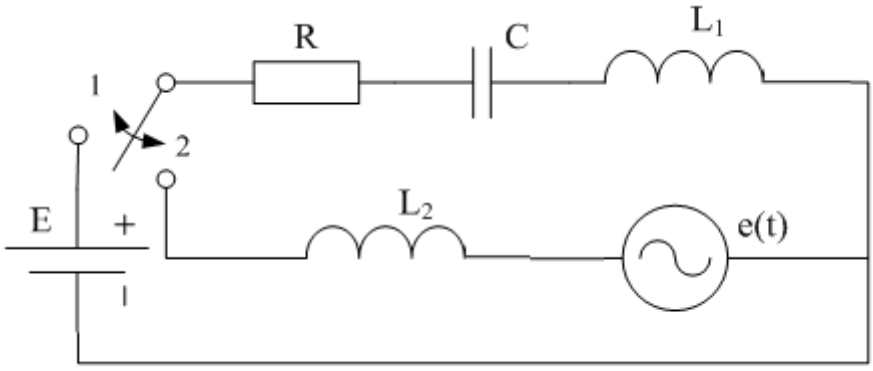
Analiza zagadnienia wymagającego rozwiązania układu równań różniczkowych wymaga w zasadzie tylko zapisania równań stanu opisujących analizowane zagadnienie.

W ćwiczeniu rozwiązywanie równań różniczkowych analizowane będzie na przykładzie stanów nieustalonych w obwodach RLC.

10.2.1. Analiza obwodu, zapisanie równań stanu

Na przykładzie obwodu pokazanego na rysunku 10.5. zaprezentowana zostanie procedura wyznaczania równań stanu i definiowania funkcji stanu dla metody

ode. W badanym przypadku analizowany będzie obwód załączany na napięcie stałe, a następnie przełączany na źródło napięcia przemiennego. Parametry analizowanego obwodu są następujące: E=100 V, e(*t*)=100sin(250*t*), R=0,25 Ω, L_1=25 mH, L_2=12 mH, C=33 mF. Załączenie obwodu następuje w chwili t=0, a przełączenie po 0,5 s.



Rys. 10.5. Schemat elektryczny analizowanego obwodu
Źródło: opracowanie własne

W przykładowym zadaniu przeanalizować i rozwiązać trzeba dwa obwody, pierwszy napięcia stałego o warunkach początkowych zerowych załączany w chwili t=0 i drugi napięcia przemiennego o warunkach początkowych określonych na podstawie wyników zadanie pierwszego.

- Etap pierwszy – obwód napięcia stałego

Proces wyznaczania równań stanu należy rozpocząć od zapisania równań Kirchhoffa opisujących analizowany przypadek. Ponieważ jest to obwód jednooczkowy zastosowanie będzie miało tylko II prawo Kirchhoffa, opisany zostanie bilans napięć [4].

$$E = R \cdot i(t) + u_{L_1} + u_C \tag{1}$$

W stanach dynamicznych w obwodach elektrycznych zależności pomiędzy prądem w obwodzie, a napięciem na kondensatorze oraz napięciem na cewce a prądem obwodu opisują zależności zapisane poniżej.

$$i(t) = C\frac{du_C}{dt}, \quad u_L = L_1\frac{di(t)}{dt} \tag{2}$$

Podstawiając zapisane zależności (2) do równanie (1), otrzymuje się równanie różniczkowe II rzędu jednej zmiennej opisujące analizowany obwód.

$$E = L_1 C \frac{d^2 u_C}{dt^2} + RC \frac{du_C}{dt} + u_C \tag{3}$$

Na podstawie równania (3) można zapisać pierwsze równanie stanu w którym występują dwie zmienne stanu u_C i $\frac{du_C}{dt}$.

$$\frac{d^2 u_C}{dt^2} = \frac{E - RC\frac{du_C}{dt} - u_C}{L_1 C} \tag{4}$$

Równanie stanu dla zmiennej u_C zapisać można na podstawie pierwszej zależności (2).

$$\frac{du_C}{dt} = \frac{i(t)}{C} \tag{5}$$

W analizowanym przypadku będzie jeszcze jedna zmienna stanu, prąd obwodu *i*. Równanie stanu dla prądu zapisać można na podstawie równanie (1) i drugiej zależności (2).

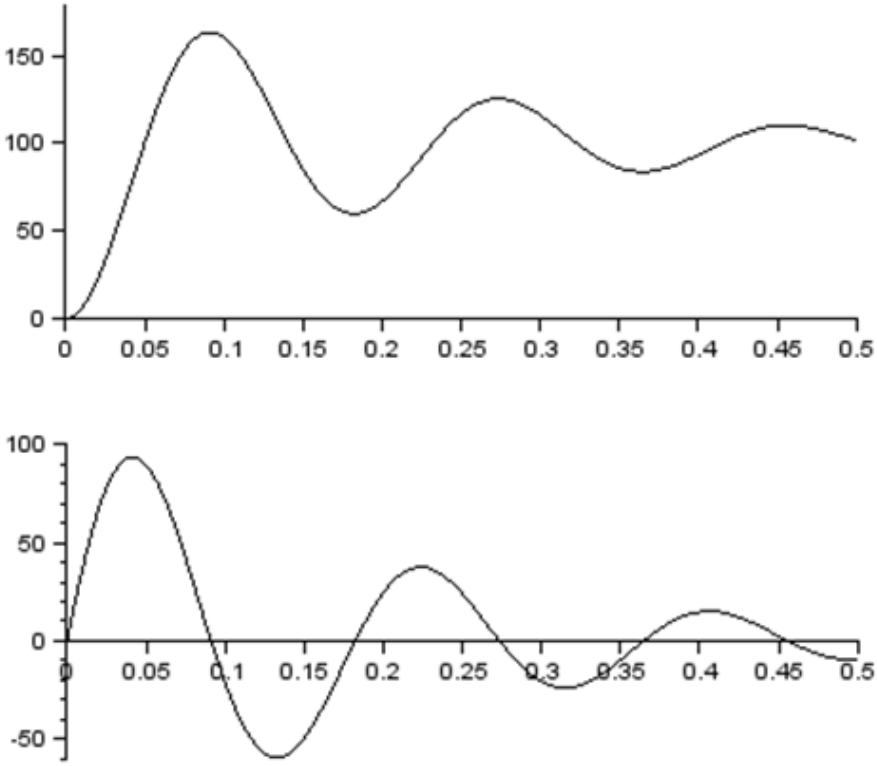
$$\begin{aligned} E &= Ri(t) + L_1 \frac{di(t)}{dt} + u_C \\ \frac{di(t)}{dt} &= \frac{E - Ri(t) - u_C}{L_1} \end{aligned} \tag{6}$$

Można więc zapisać funkcję opisującą analizowany obwód. Funkcja będzie zawierała układ trzech równań trzech zmiennych stanu.

$$x(1) = \frac{du_C}{dt}, \quad x(2) = u_C, \quad x(3) = i(t) \tag{7}$$

```
function [y]=stany(t,x)
      y(1)=(E-R*C*x(1)-x(2))/(L1*C);
      y(2)=x(3)/C;
      y(3)=(E-R*x(3)-x(2))/L1;
endfunction
```

Zapisana funkcję można już podpiąć do funkcji ode u rozwiązać zadanie. Rozwiązaniem będzie macierz trójwierszowa zawierająca przebieg zmian trzech zmiennych stanu w funkcji czasu, jak pokazano na rysunku 10.6.



Rys. 10.6. Przebiegi napięcia na kondensatorze i prądu w obwodzie po załączeniu włącznika w pozycję 1
Źródło: opracowanie własne

Warunki początkowe do drugiego etapu zadania wynoszą odpowiednio dla kolejnych zmiennych stanu: $\frac{du_C}{dt}$ =-288,3947, $u_C(0.5)$= 101,61175, i(0.5)= -9,517025.

- Etap drugi – obwód napięcia przemiennego

W drugim etapie analizy zadania postępuje się analogicznie, wychodząc od równanie bilansu napięć analizowanego obwodu.

$$e(t) = R \cdot i(t) + u_{L_1} + u_C + u_{L_2} \tag{8}$$

Korzystając z równań (2) przekształca się równanie (8) do postaci równania różniczkowego jednej zmiennej.

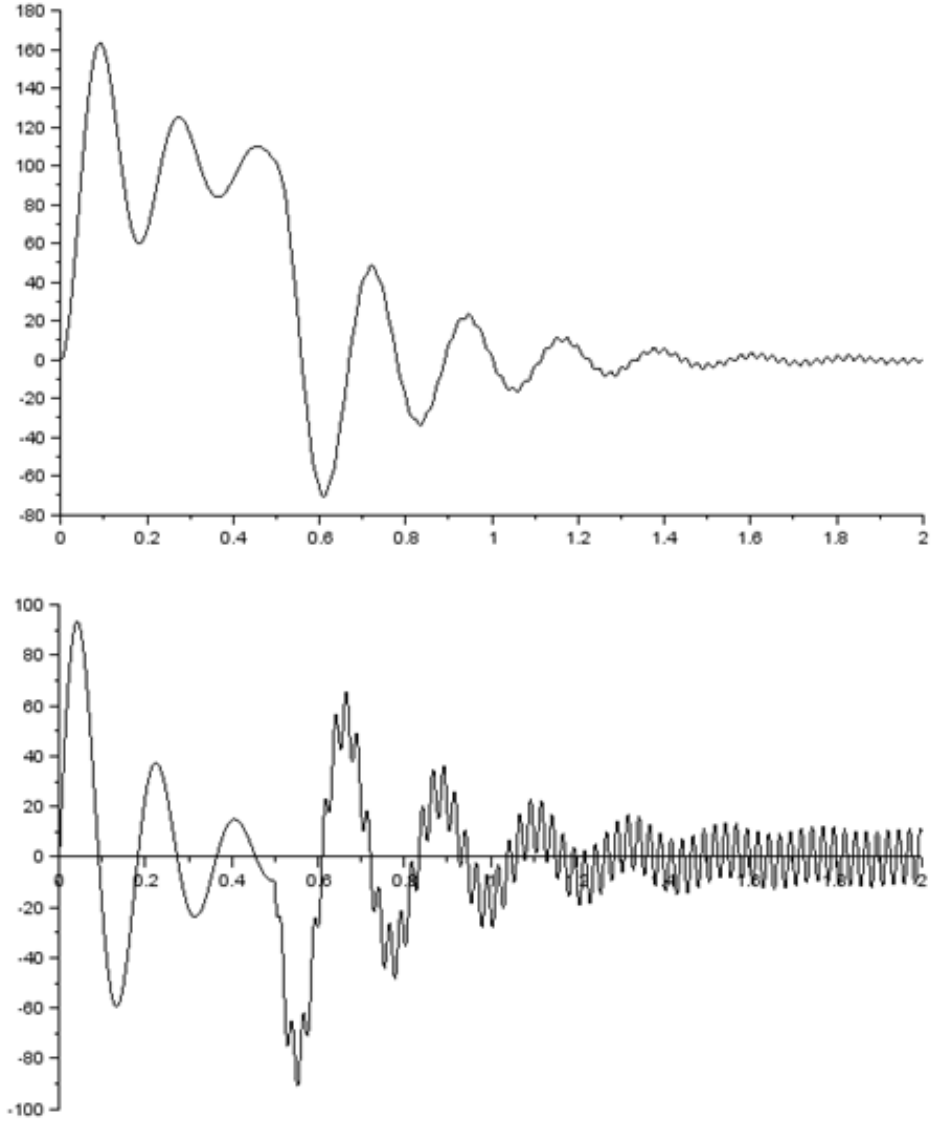
$$e(t) = RC\frac{du_c}{dt} + L_1\frac{di(t)}{dt} + L_2\frac{di(t)}{dt} + u_C$$
$$e(t) = RC\frac{du_c}{dt} + (L_1 + L_2)\frac{di(t)}{dt} + u_C \quad (9)$$

$$e(t) = (L_1 + L_2)C\frac{d^2u_C}{dt^2} + RC\frac{du_C}{dt} + u_C \quad (10)$$

Dalej postępowanie jest analogiczne jak w etapie pierwszym i w efekcie zapisuje się funkcję równań stanu. Dodatkowym elementem jest konieczność zdefiniowania funkcji opisującej przebieg napięcia źródła i podpięcie jej do funkcji równań stanu [2].

```
function [y]=stany_2(t,x)
      y(1)=(Et(t)-R*C*x(1)-x(2))/((L1+L2)*C);
      y(2)=x(3)/C;
      y(3)=(Et(t)-R*x(3)-x(2))/(L1+L2);
endfunction
funkction y=Et(t)
      y=Emax*sin(300*t);
endfunction
```

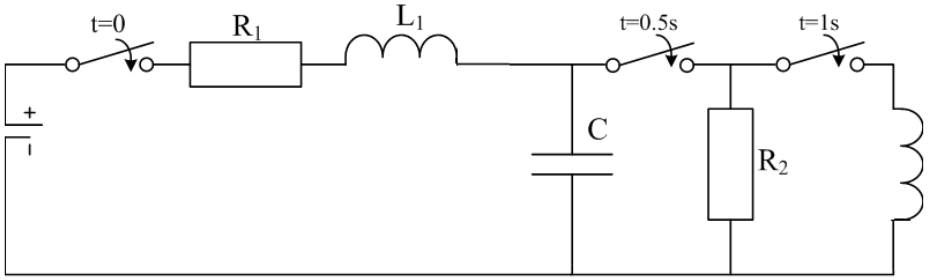
Drugi etap liczony jest w czasie od chwili przełączenia 0,5 s do wyłączenia obwodu w chwili t=2 s. Łącząc wyniki uzyskane z rozwiązania obu układów równań uzyskuje się przebiegi napięcia na kondensatorze i prądu w analizowanym obwodzie, jak pokazano na rysunku 10.7.



Rys. 10.7. Przebiegi napięcia na kondensatorze i prądu w obwodzie
Źródło: opracowanie własne

Zadanie

W oparciu o zamieszczony powyżej przykład przeprowadzić symulację obwodu załączanego kaskadowo, pokazanego na rysunku 10.8. lub zadanego przez prowadzącego ćwiczenia.



Rys. 10.8. Schemat obwodu do rozwiązania w zadaniu

Parametry przykładowego obwodu są następujące: E=100 V, L_1=0,5·L_2=150 μH, C=0,5 mF, R_2=100·R_1, R_1 należy dobrać tak, aby po załączeniu włącznika w chwili t=0 przebieg prądu był oscylacyjny. Dobraną wartość rezystancji należy potwierdzić przeprowadzonym dowodem rachunkowym.

W zadaniu należy wykreślić przebiegi prądów w obwodzie oraz napięć na elementach biernych.

Literatura

[1] Brozi A., *Scilab w przykładach,* Poznań, Nakom, 2010

[2] *Strona Scilab Enterprises S.A.S. [online]*, [dostęp 23 marca 2013], Dostępny w World Wide Web: http://www.scilab.org

[3] Lachowicza C. T., Matlab, Scilab, Maxima. Opis i przykłady zastosowań, wydawnictwo Politechniki Opolskiej, 2005

[4] Majchrzak E., Mochnacki B., *Metody numeryczne. Podstawy teoretyczne, aspekty praktyczne i algorytmy*, Wydawnictwo Politechniki Śląskiej, Wydanie IV, 2004

[5] Krakowski M., *Elektrotechnika teoretyczna- pole ektromagnetyczne,* Tom 2, PWN, 1995

[6] Bolkowski S., *Elektrotechnika*, WSiP, 2012

[7] Povstenko J., *Wprowadzenie do metod numerycznych,* Akademicka Oficyna wydawnicza EXIT, Warszawa 2002

[8] Bolkowski S., *Elektrotechnika teoretyczna*, Tom I, WNT, Warszawa 1986

[9] Wbudowany podręcznik programowania (help) SciLab